

Spis Treści

- [1. Wstęp](#)
- [2. Cele i założenia](#)
- [3. Przebieg pracy](#)
- [4. Część informatyczna](#)
 - [4.1. Backend - strona serwerowa](#)
 - [4.2. Frontend - strona użytkownika](#)
 - [4.3. Działanie aplikacji](#)
 - [4.4. Wygląd aplikacji](#)
- [5. Część mechatroniczna - sterownik](#)
 - [5.1. Prototyp sterownika](#)
 - [5.2. Gotowy produkt - sprzęt](#)
 - [5.3. Gotowy produkt - program](#)

1. Wstęp

2. Cele i założenia

3. Przebieg pracy

4. Część informatyczna

Jedną z najważniejszych zalet **Net-Worku** jest jego infrastruktura informatyczna. W celu zbudowania funkcjonującego systemu wykorzystane zostały najnowsze frameworki oraz języki programowania. Samo oprogramowanie zostało napisane w kilku językach, strona serwerowa została napisana w języku **Go**, natomiast strona użytkownika została napisana w języku **JavaScript** wraz z frameworkiem **Vue.js** i komponentami w najnowszym standardzie material, **Vuetify**, do automatyzacji komplikacji i uruchamiania oprogramowania został wykorzystany **Make** oraz **Bash**.

Komunikacja w oprogramowaniu opiera się na protokołach **Message Queuing Telemetry Transport (MQTT) w wersji 3.11** oraz **Representational State Transfer (REST)**, oba protokoły są przydatne w różnych kontekstach komunikacyjnych. Protokół **MQTT** został wykorzystany ze względu na jego lekkość i efektywność, co czyni go idealnym rozwiązaniem dla aplikacji związanych z **Internetem Rzeczy (IoT)** oraz systemów o niskich wymaganiach zasobowych. Dzięki modelowi publikacji i subskrypcji (**PUBSUB**), **MQTT** umożliwia efektywną wymianę danych między urządzeniami w czasie rzeczywistym. Z kolei architektura **REST** stanowi uniwersalny interfejs komunikacyjny, który pozwala na zarządzanie zasobami w sposób zrozumiały dla ludzi oraz maszyn. Wykorzystanie tych protokołów pozwala na elastyczne i skalowalne budowanie aplikacji, które są w stanie efektywnie komunikować się z różnymi systemami i urządzeniami.

4.1. Backend - strona serwerowa

W celu zbudowania bezpiecznego oraz wydajnego serwera **REST** i brokera **MQTT**, wykorzystane zostały następujące pakiety języka **Go**:

- **GIN 1.9.1**
- **COMMQTT 2.5.4**

Technologie te pozwalają na bezpośrednią integrację obu protokołów komunikacyjnych w jednej bazie kodu. W przypadku pakietu **GIN** dodatkowe możliwości wprowadzania **middleware'ów** (programów pośrednich między żądaniem a właściwą częścią aplikacji) pozwolity na proste wbudowanie dodatkowych zabezpieczeń dostępu do serwera **REST**, takich jak:

- **Ograniczenie szybkości zapytań (Rate Limiting)** które jak sama nazwa wskazuje zmniejsza częstotliwość odpowiedzi na zapytania pochodzących od jednego klienta w określonym przedziale czasowym, co może pomóc w zapobieganiu nadmiernemu obciążeniu serwera.
- **Tokeny JWT (Json Web Token)** które stanowią sposób na uwierzytelnianie i autoryzację użytkowników w serwerze **REST**. **JWT** są tokenami zawierającymi informacje o użytkowniku oraz jego uprawnieniach, podpisane przez serwer, co pozwala na bezpieczne przesyłanie tych danych między klientem a serwerem. Dzięki nim można łatwo kontrolować dostęp do zasobów oraz identyfikować użytkowników w systemie.

Natomiast pakiet **COMMQTT** zapewnia nie tylko implementację protokołu **MQTT**, ale również możliwość bezpośredniego konfigurowania, monitorowania oraz ingerencji w broker **MQTT**. Dzięki temu można skonfigurować różne parametry działania brokera w sposób programowy, takie jak na przykład maksymalny rozmiar wiadomości czy maksymalna liczba połączonych klientów, a także monitorować jego wydajność i obciążenie. Ważną kwestią jest też możliwość uwierzelniania klientów którzy próbują połączyć się z brokerem, realizowane jest to poprzez pozyskiwanie danych na temat klientów z bazy danych oraz porównywanie danych z którymi dany klient próbuje się połączyć. W wyniku tego można zapewnić bezpieczne i kontrolowane połączenia między klientami a brokerem **MQTT**.

Dzięki wykorzystaniu tych pakietów możliwe było zrealizowanie nie tylko bezpiecznego, ale także wydajnego serwera **REST** oraz brokera **MQTT**, który spełnia wymagania zarówno pod kątem funkcjonalności, jak i wydajności.

Ważną rzeczą w naszym oprogramowaniu jest również integracja z relacyjną **bazą danych PostgreSQL**, aplikacja wykorzystuje ją aby przetrzymywać lub odczytywać dane na temat zarejestrowanych użytkowników, urządzeń, konfiguracji urządzeń czy pluginów. Ingerować w nią mogą zarejestrowani użytkownicy posiadający odpowiedni poziom uprawnień.

Dodatkowe narzędzia i paczki użyte przy tworzeniu strony serwerowej:

- Curl
- Postman
- MQTT Explorer
- PostgreSQL Explorer
- github.com/charmbracelet/lipgloss
- github.com/didip/tollbooth
- github.com/gin-contrib/cors
- github.com/gin-gonic/gin
- github.com/golang-jwt/jwt/v5
- github.com/hashicorp/mdns
- github.com/joho/godotenv
- github.com/spf13/viper
- golang.org/x/crypto
- gorm.io/gorm

4.2. Frontend - strona użytkownika

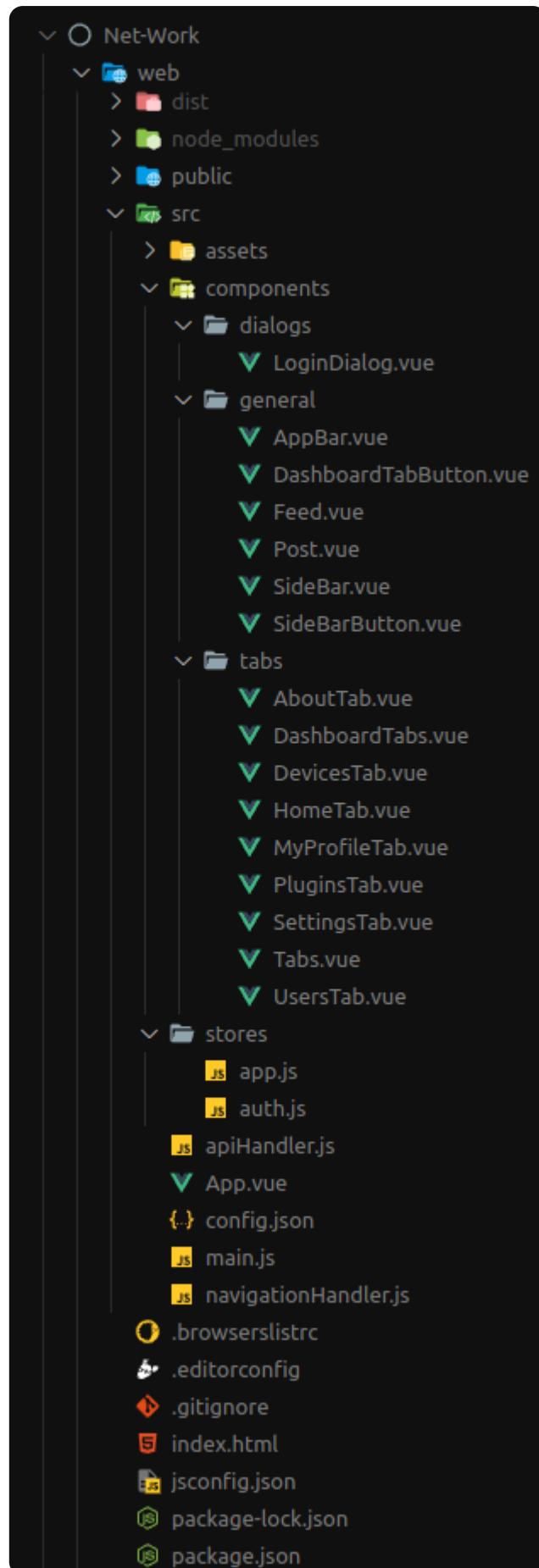
Aplikacja internetowa w całości oparta została na frameworku **Vue.js 3**. Dzięki zastosowaniu tej technologii aplikacja pod względem wydajnościowym wyraźnie wyprzedza inne projekty, które z domyślu oparte są o statyczne strony internetowe.

Aplikacja internetowa oparta jest na nowoczesnych technologiach takich jak framework **Vue.js 3**, **Vite 5.2.2** jako narzędzie do budowania projektu w jedna spójną całość gotową do uruchomienia, oraz **Vuetify**, biblioteki komponentów **Material Design**. W połączeniu z biblioteką **Pinia 2.1.7** do zarządzania stanem aplikacji oraz **Axios 1.6.7** do komunikacji z serwerem, te technologie pozwoliły nam na stworzenie wyjątkowo wydajnej aplikacji jednostronnej **SPA (Single Page Application)**. **Vue.js** umożliwia dynamiczne routowanie po stronie użytkownika, eliminując jednocześnie konieczność przeładowywania całej strony podczas przejść miedzy różnymi widokami, co poprawia doświadczenie i komfort użytkownika. **Pinia** wraz z **Axios** zapewniają bezpieczną i wydajną komunikację z serwerem, a wykorzystanie komponentów z **Vuetify** ułatwiło stworzenie interfejsu użytkownika zgodnego ze standardem **Material Design**. Dzięki temu, architektura kodu aplikacji staje się bardziej przejrzysta i skalowalna, co znacznie ułatwia rozwój aplikacji o nowe funkcjonalności.

Przykładowy komponent

```
1 <template>
2   <v-window-item :value="value" v-model="selectedSubTabName">
3     <v-tabs
4       v-model="selectedSubTabName"
5       align-tabs="center"
6       color="deeper-purple-accent-4"
7       mandatory
8       stacked
9     >
10    <slot name="buttons" />
11  </v-tabs>
12
13  <v-window
14    v-model="selectedSubTabName"
15    direction="horizontal"
16    elevation="0"
17    mandatory
18  >
19    <slot name="content" />
20  </v-window>
21 </v-window-item>
22 </template>
23
24 <script>
25 export default {
26   name: 'DashboardTab',
27   props: {
28     value: {
29       type: Object,
30       required: true,
31     },
32     childValue: {
33       type: Object,
34       required: true,
35     },
36   },
37   computed: {
38     selectedSubTabName: {
39       get() {
40         console.log('getting dashboard tab name ', this.value);
41         return this.childValue;
42       },
43       set(newValue) {
44         if (this.childValue && newValue !== this.childValue) {
45           console.log('updating dashboard tab name ', newValue);
46           this.$emit('update:childValue', newValue );
47         }
48       },
49     },
50   },
51   mounted() {
52     console.log('DashboardTab component received value and childValue props:', this.value, this.childValue);
53   }
54 };
55 </script>
```

Struktura plików frontendu

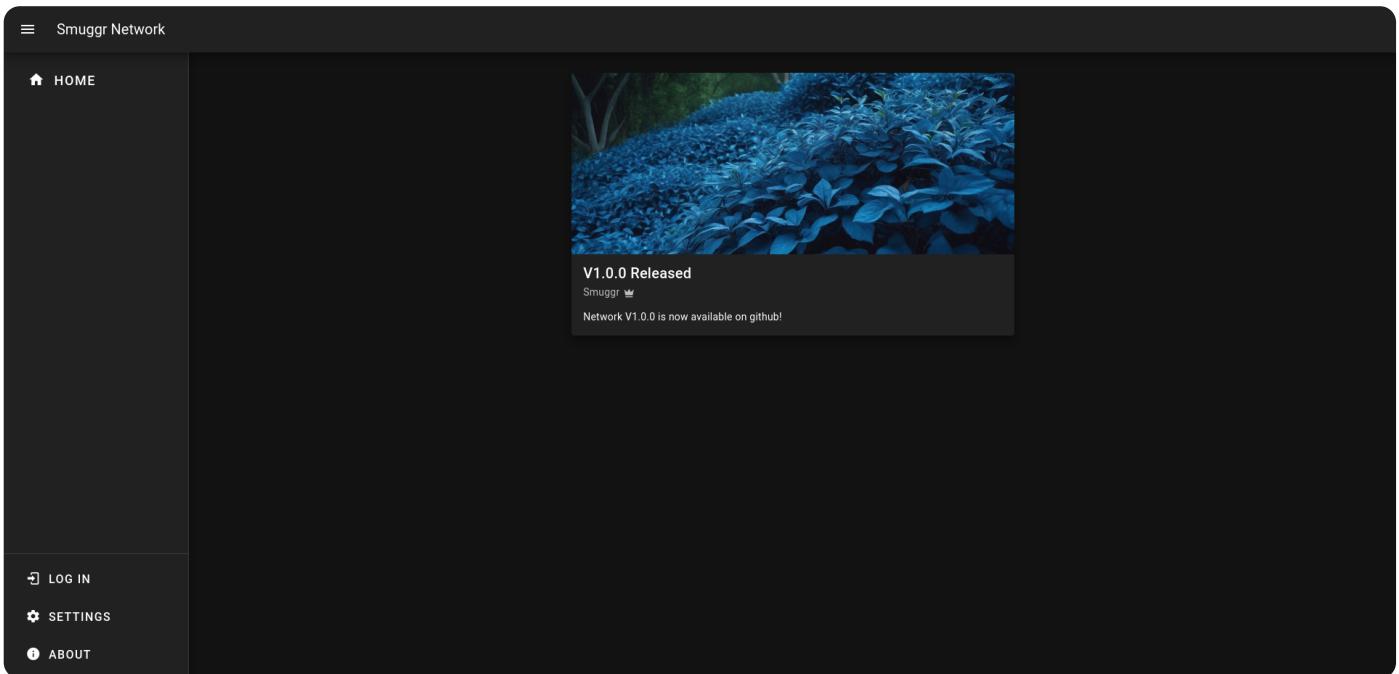


4.3. Działanie aplikacji

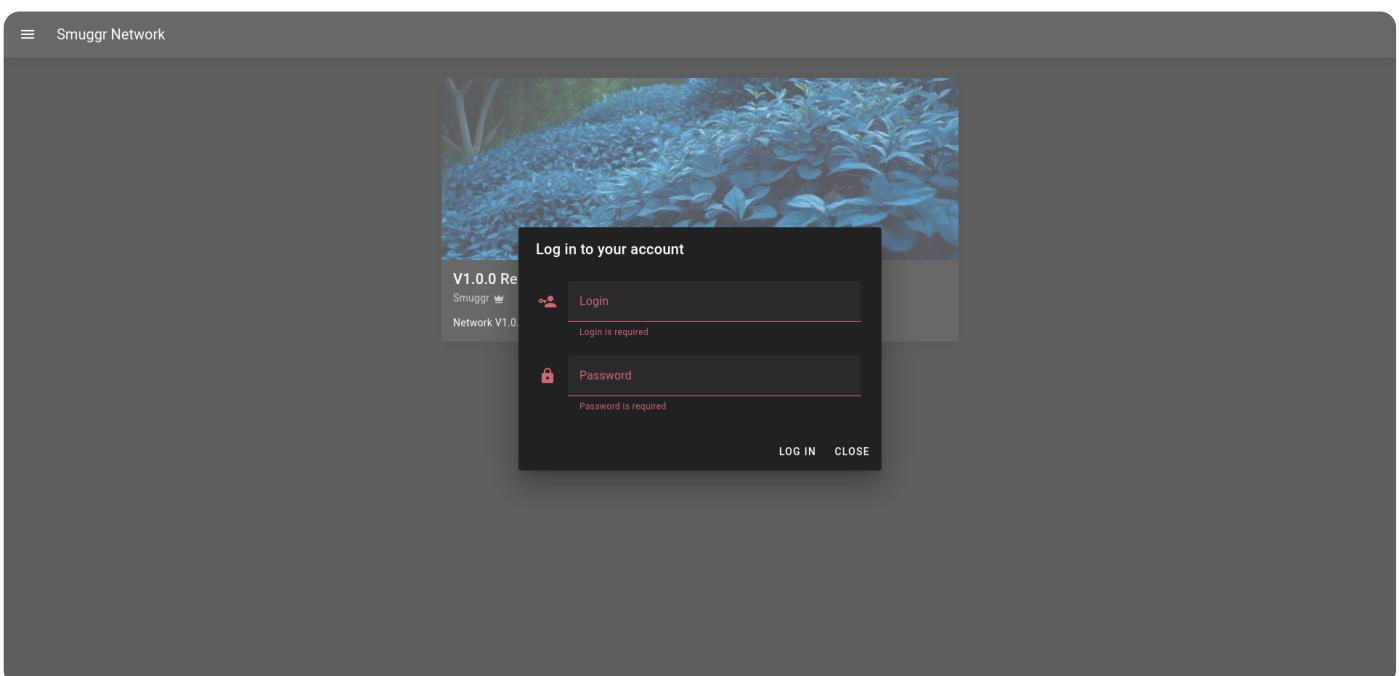
Głównymi zadaniami naszego oprogramowania **Net-Work** jest przedewszystkim łączenie urządzeń w jednym miejscu i umożliwienie niekoniecznie technologicznie zaawansowanym użytkownikom korzystanie z tych urządzeń. Urządzenia mogą łączyć się z **brokerem MQTT** na kilka sposobów:

4.4. Wygląd aplikacji

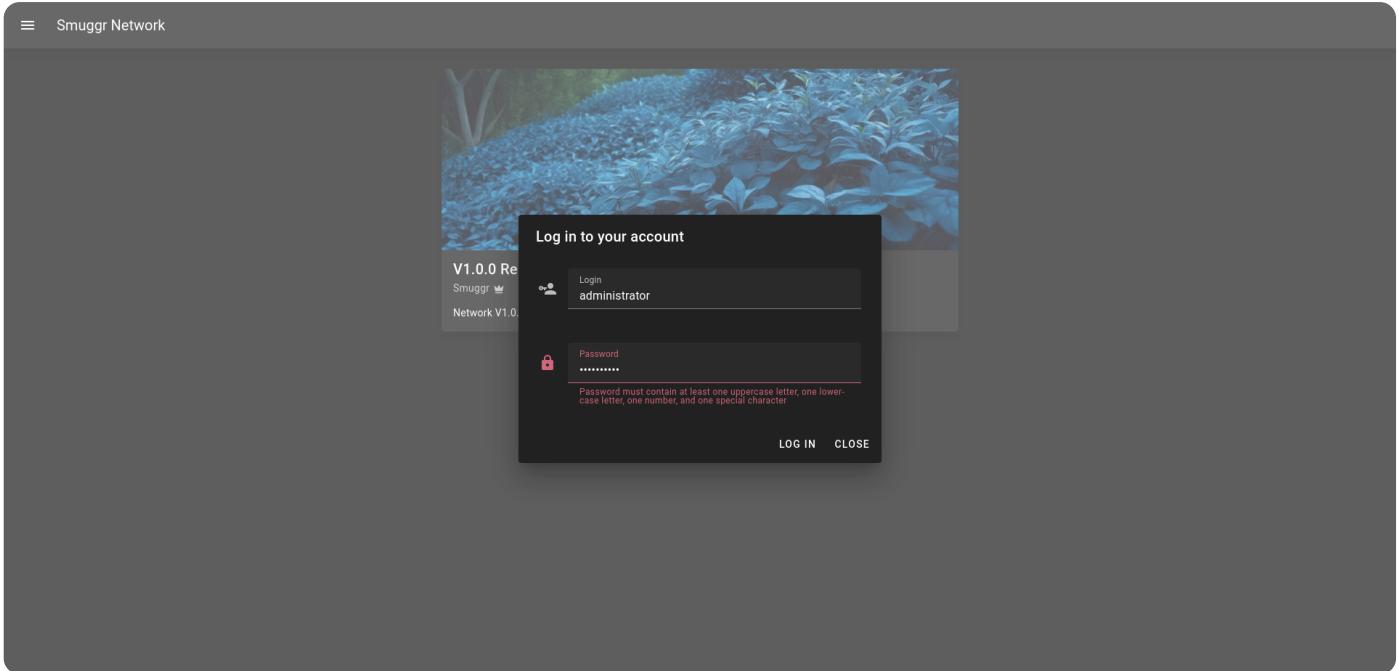
Strona główna przed zalogowaniem



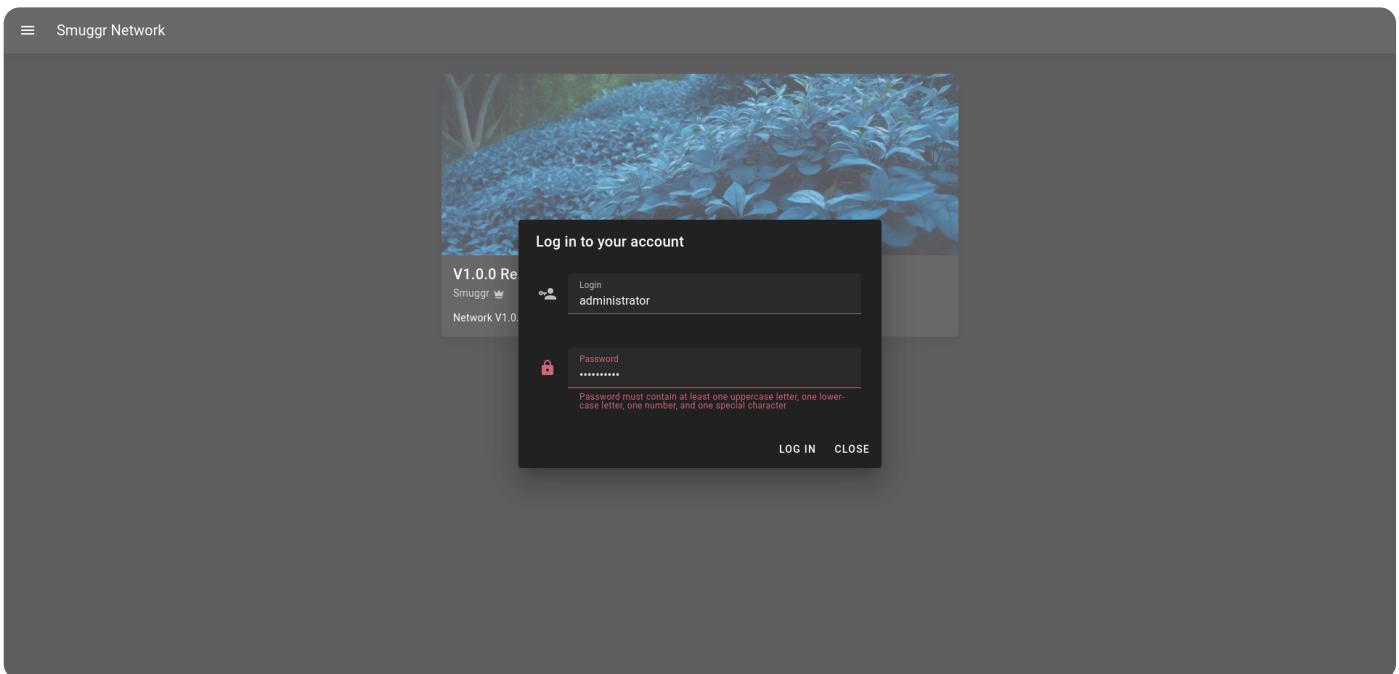
Okno logowania



Demonstracja weryfikacji danych



Demonstracja weryfikacji danych



Strona główna po zalogowaniu

☰ Smuggr Network

- HOME
- DASHBOARD

MY PROFILE

LOG OUT

SETTINGS

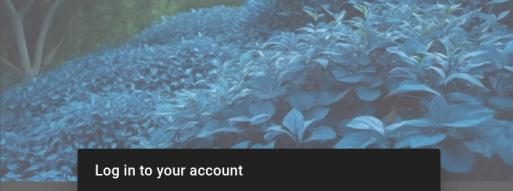
ABOUT



V1.0.0 Released
Smuggr 🌟
Network V1.0.0 is now available on github!

Profil zalogowanego użytkownika

☰ Smuggr Network



V1.0.0 Re
Smuggr 🌟
Network V1.0.

Log in to your account

Login

Login is required

Password

Password is required

LOG IN CLOSE

Strona "O stronie"

The screenshot shows the 'About' section of the Smuggr Network website. On the left sidebar, there are links for HOME, DASHBOARD, MY PROFILE, LOG OUT, SETTINGS, and ABOUT. The main content area is titled 'Source Code' and contains a button labeled 'Smuggr Network on GitHub' with a GitHub icon.

Panel z urządzeniami

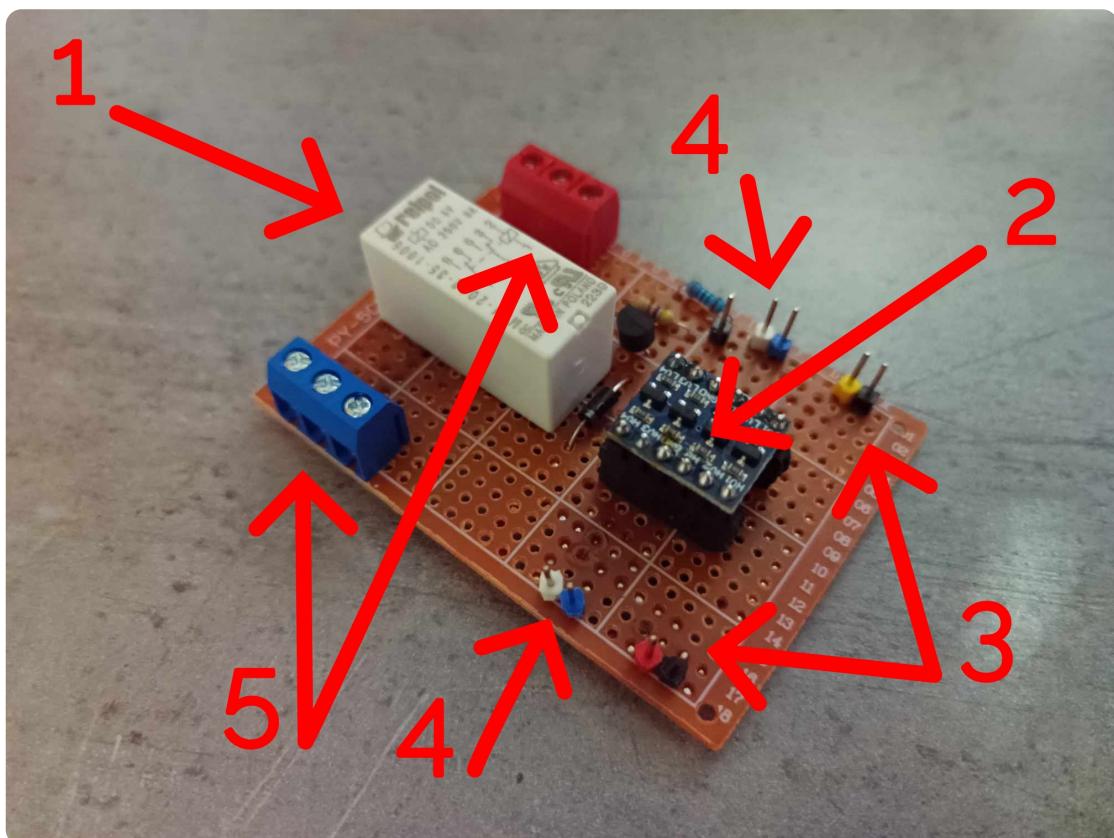
The screenshot shows the 'Devices' section of the Smuggr Network website. On the left sidebar, there are links for HOME, DASHBOARD, MY PROFILE, LOG OUT, SETTINGS, and ABOUT. The main content area features a 'DEVICES' tab, which is currently selected, indicated by a blue underline. Below it, there's a card titled 'Schedule Keepr' showing a single scheduled task. At the bottom right of the card are two buttons: 'RELAY OFF' and 'RELAY ON'.

5. Część mechatroniczna - sterownik

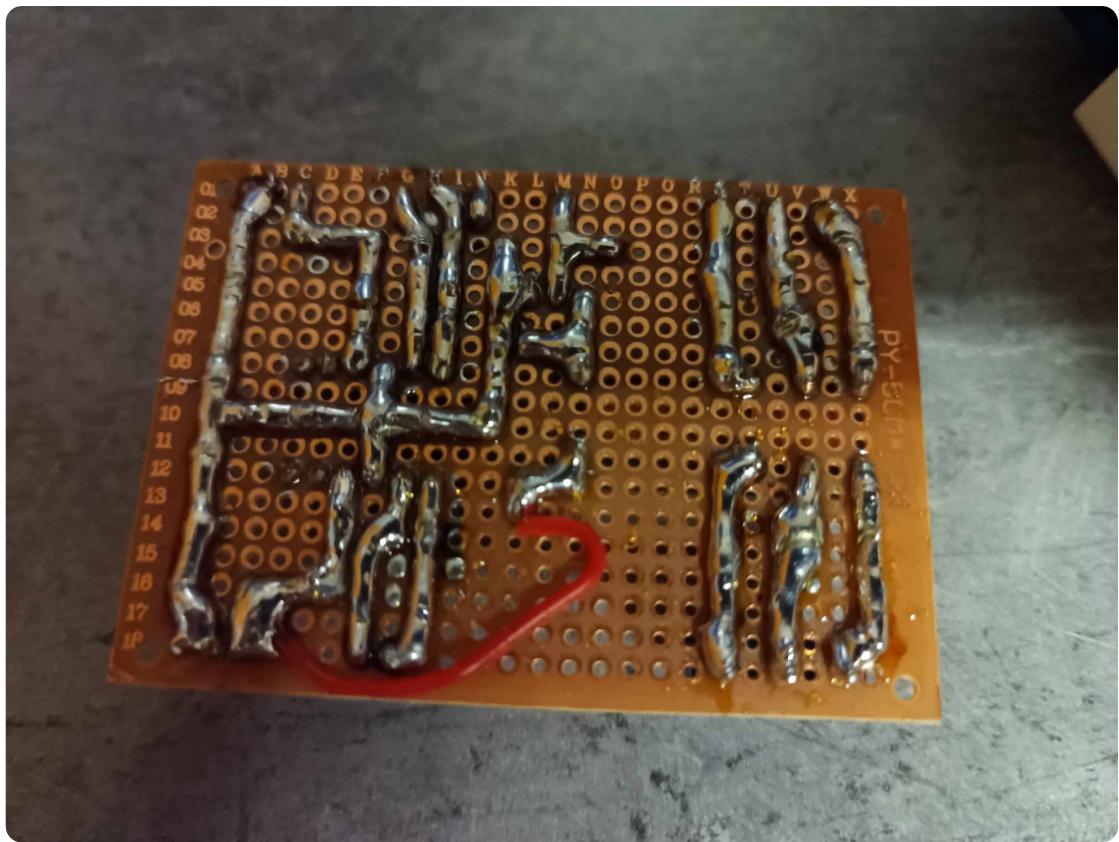
Naszym projektem nie jest jedynie oprogramowanie, należy do niego również nasz sterownik **Schedule-Keepr 1.0** który jest jednocześnie pierwszym urządzeniem funkcjonującym w naszym systemie. Jego zadaniem jest automatyzowanie funkcji aktywacji (w odpowiednim przedziale czasowym lub na żądanie) dzwonków lub jakiegokolwiek innego peryferium które może być sterowane wyjściem przekaźnikowym.

5.1. Prototyp sterownika

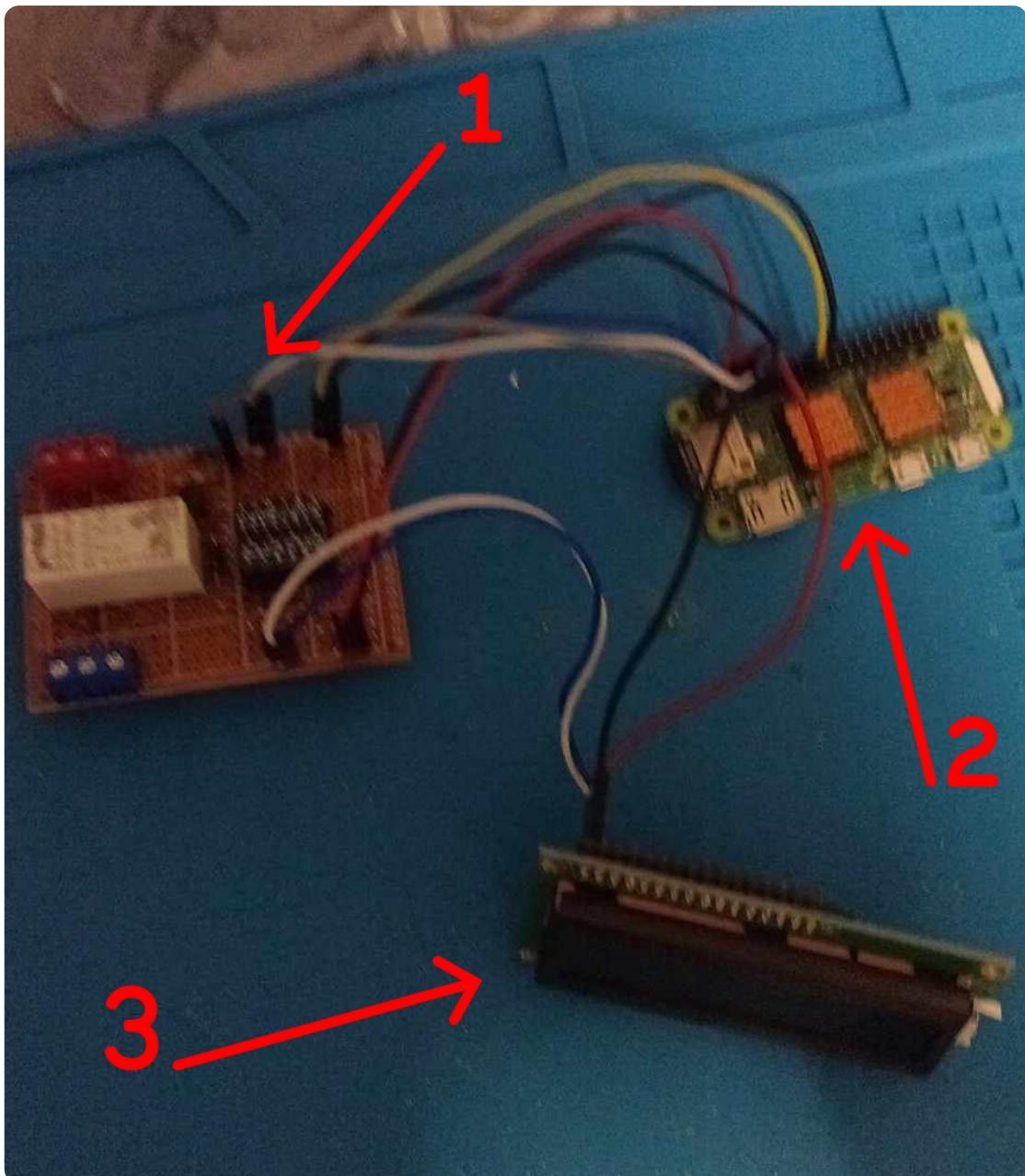
Prototyp PCB



1. Przekaźnik mechaniczny wraz z układem sterowania
2. Konwerter poziomów logicznych
3. Zasilanie 5V oraz 3.3V
4. Linie I2C o napięciu logicznym 5V oraz 3.3V
5. Złącza śrubowe przekaźnika



Opis tymczasowego złożenia elementów



1. Prototyp PCB

2. Raspberry Pi Zero W 2

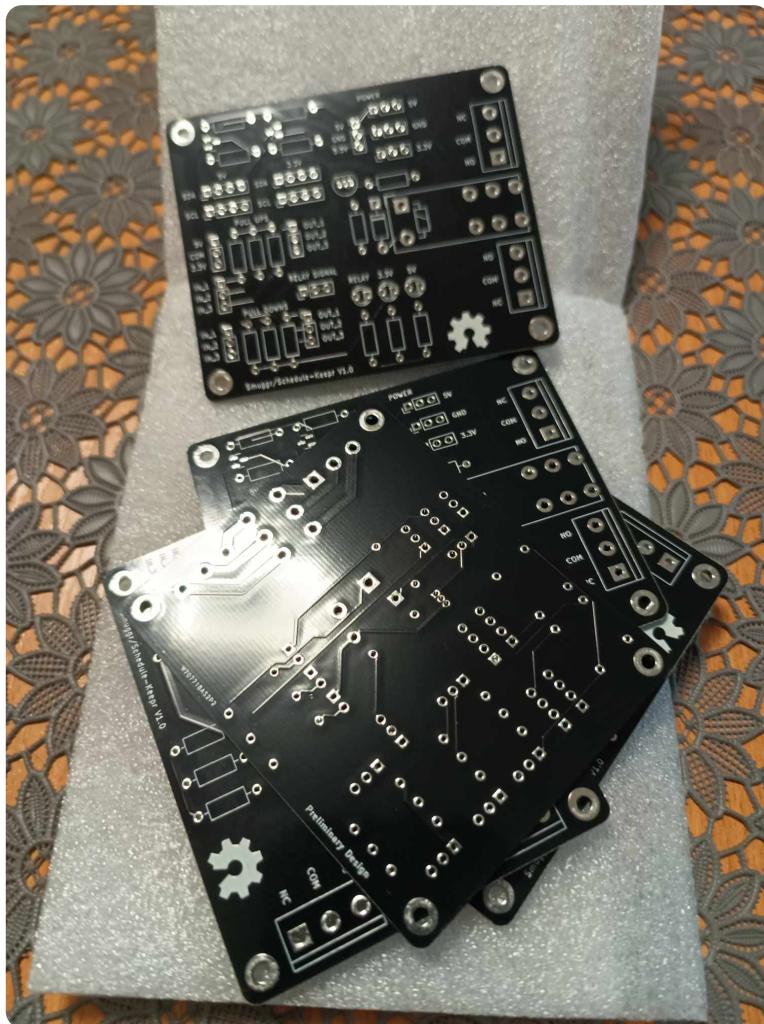
3. Wyświetlacz LCD

W prototypie naszego sterownika zdecydowaliśmy się tymczasowo zastosować popularny komputer jednopłytkowy Raspberry Pi zero W 2, stanowił on jednostkę centralną która wykonywała dedykowany program. Dodatkowo, wykorzystaliśmy wyświetlacz LCD, który umożliwiał intuicyjne wyświetlanie informacji użytkownikowi oraz poprawiał interakcję z нашym systemem. Prototyp, jak widać na załączonym zdjęciu, nie posiadał na początku żadnej obudowy.

5.2. Gotowy produkt - sprzęt

W gotowym produkcie zamiast płytka stykowej lub perforowanej - przydatnych w pierwszych fazach budowy i testowania sterownika - została stworzona dedykowana płytka PCB, którą stosuje się praktycznie we wszystkich profesjonalnych urządzeniach elektronicznych. Wynika to między innymi z tego że płytki PCB świetnie nadają się do tworzenia dowolnych układów elektronicznych o dowolnej złożoności.

PCB od razu po rozpakowaniu



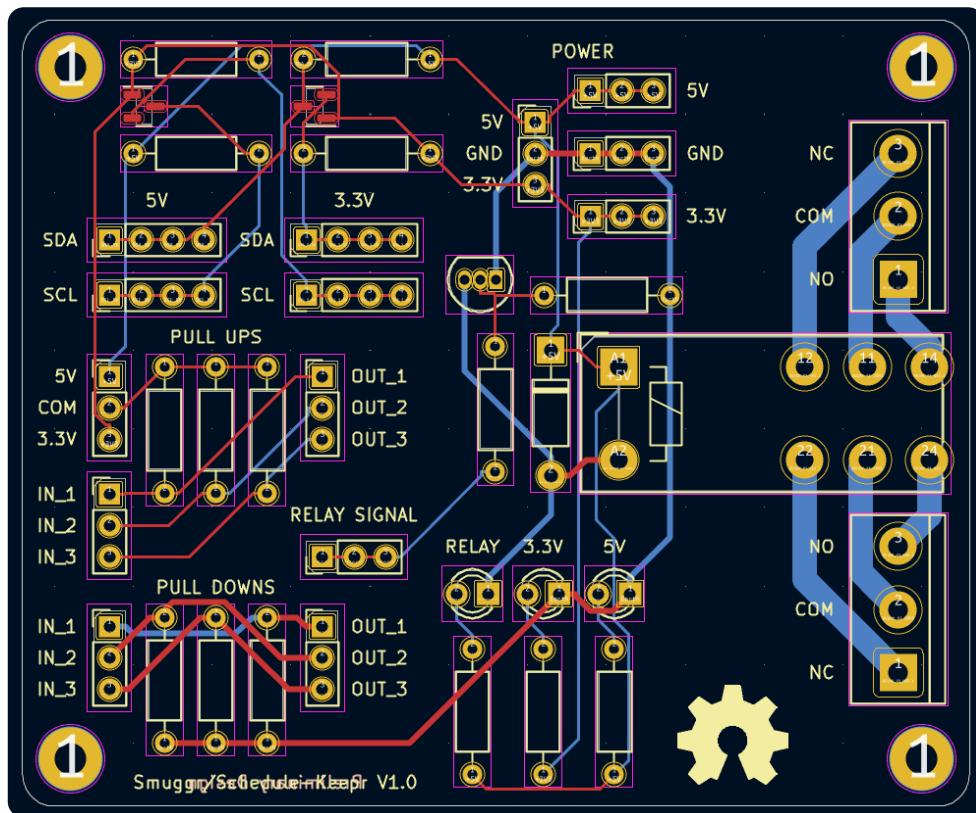
Do zaprojektowania schematu jak i układu płytki PCB sterownika wykorzystany został program **KiCad 6.0.2**, jest to bardzo popularny wybór wśród entuzjastów elektroniki jak i profesjonalistów. Program ten oferuje zaawansowane narzędzia do projektowania schematów i układów PCB, co pozwoliło efektywnie stworzyć projekt sterownika. Jego popularność wynika z tego, że jest darmowy i otwarty źródłowy, co czyni go dostępnym dla szerokiego grona użytkowników.

*Zamówienie wyprodukowania płytEK (10 sztuk) zostało złożone na stronie **PCBWay**.*

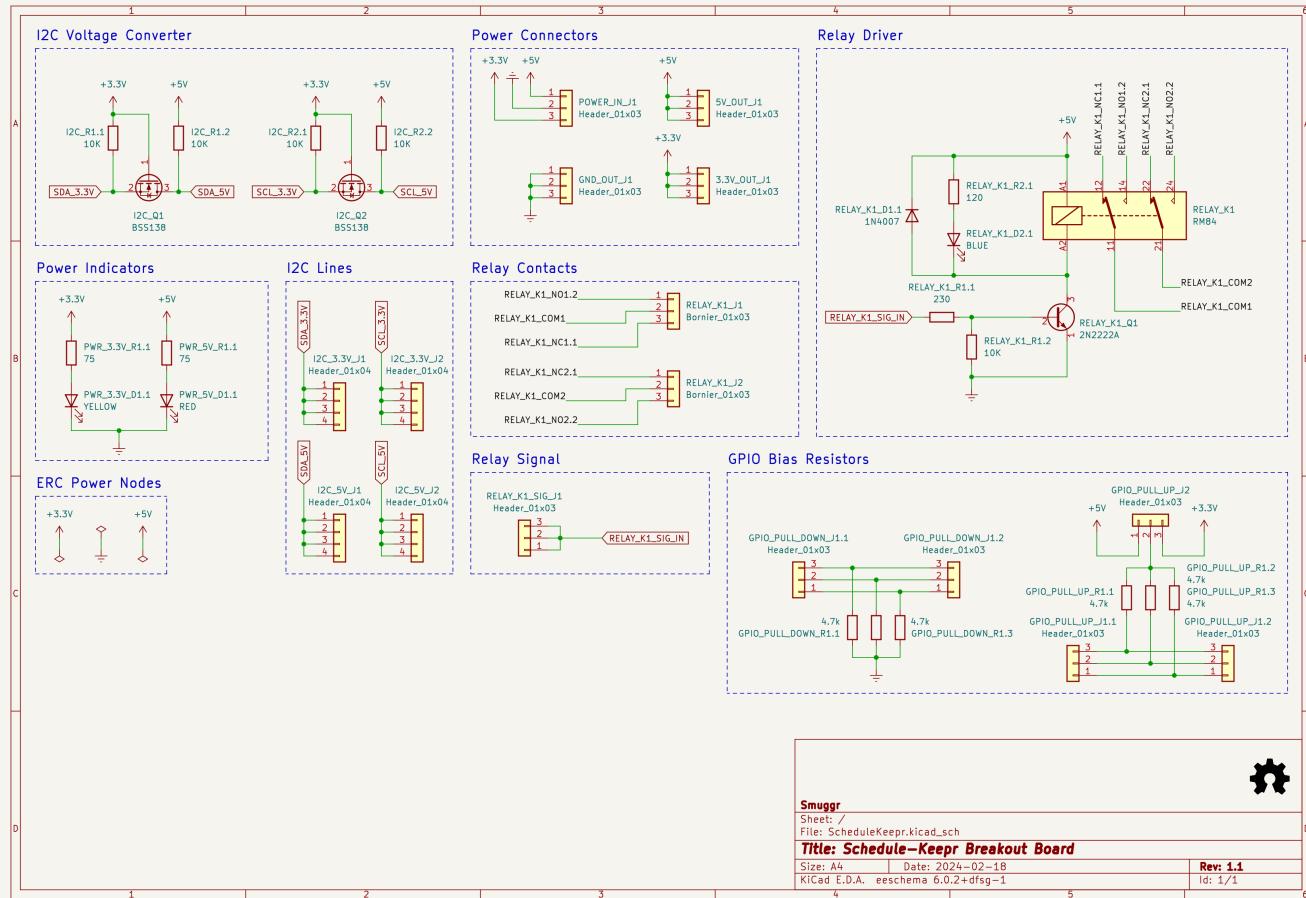
Zamówiona płytka tuż po zrealizowaniu

Product	Product Action	Order Status	Order Action
Order ID: G1204787 Delivery Address Order time & date : 2024/2/22 8:46:23	Service: Kingsley Share&Sell Production Tracking Repeat Order Open Dispute	Order Completed View Detail View Logistics	Order amount: \$ 32.40
 View 76.2x63.5mm 2 Layers, Thickness:1.6 mm, Finished copper:1, Surface finish:HASL with lead [Product No.: W707718AS2P2]			Invoice Leave Feedback / Review

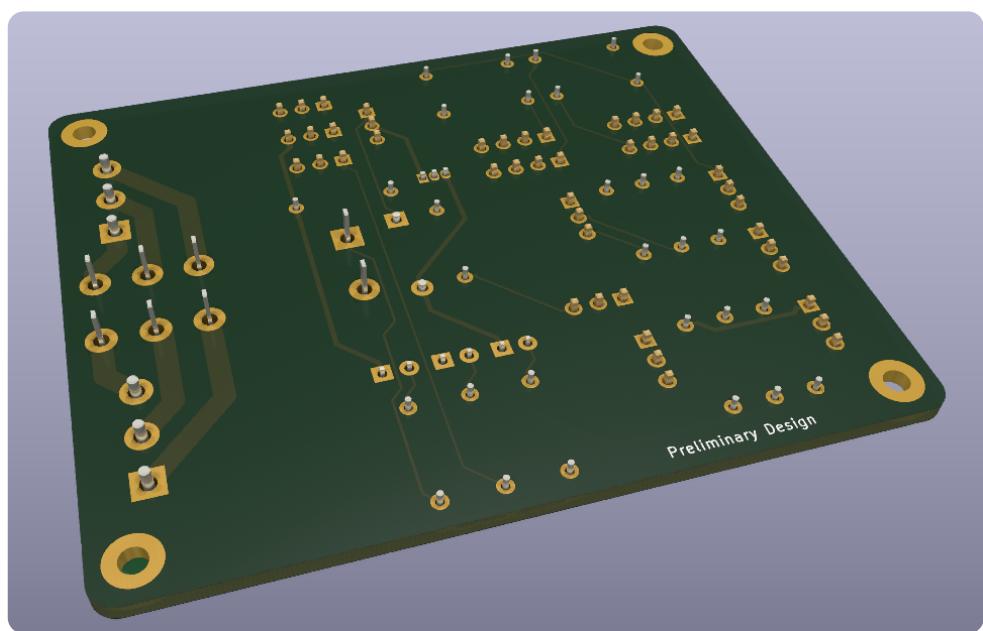
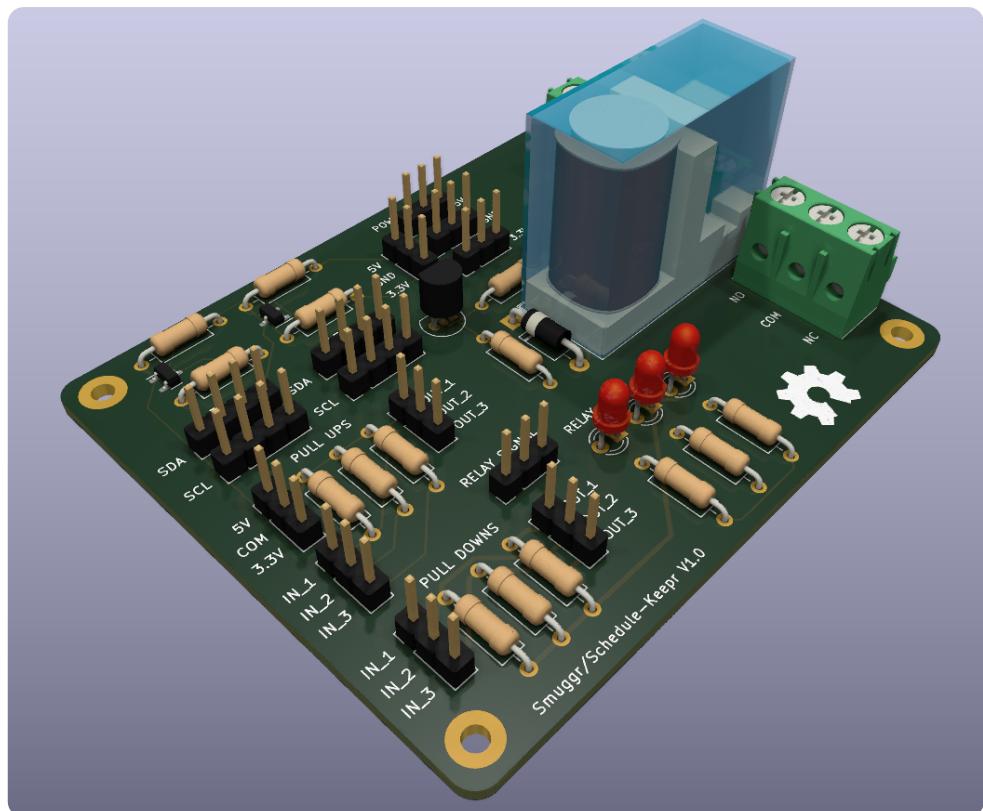
Układ elementów PCB



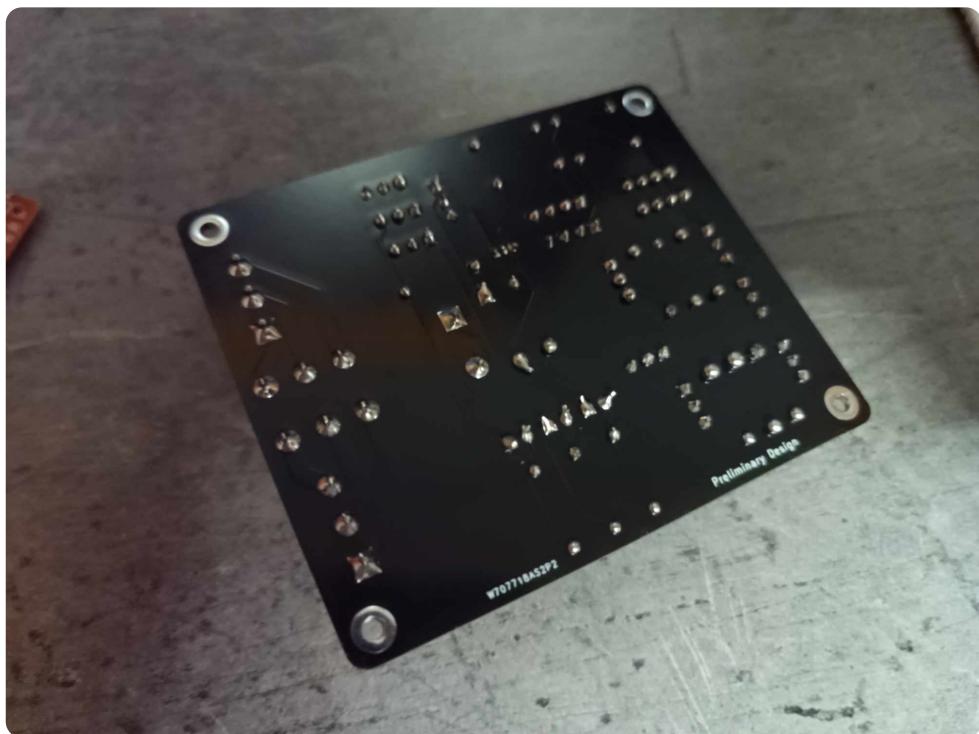
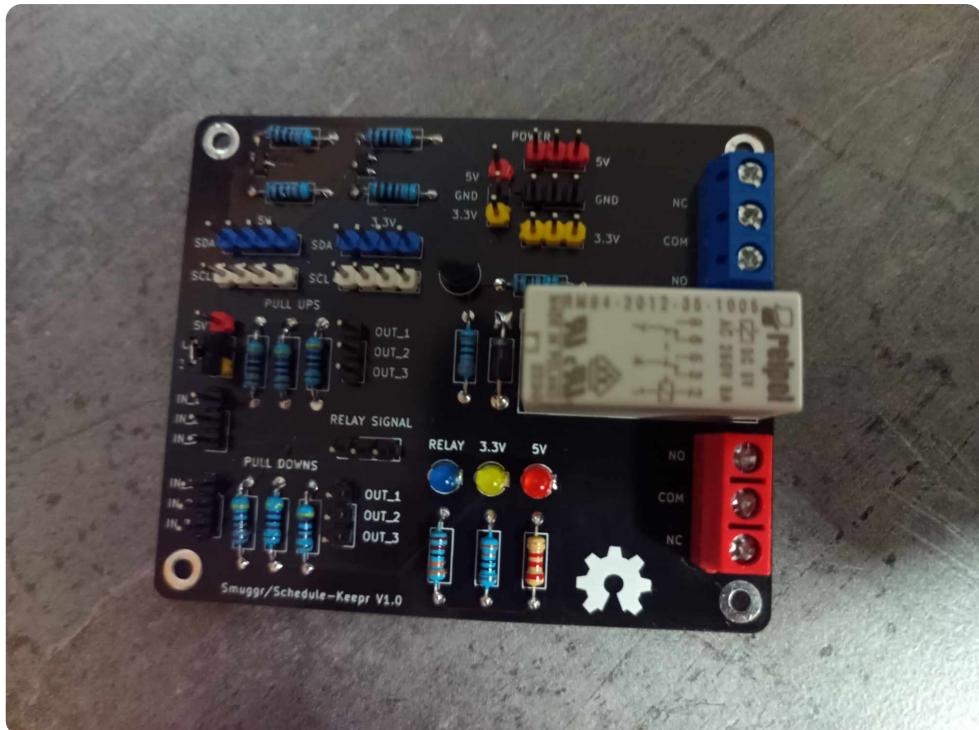
Schemat elektryczny



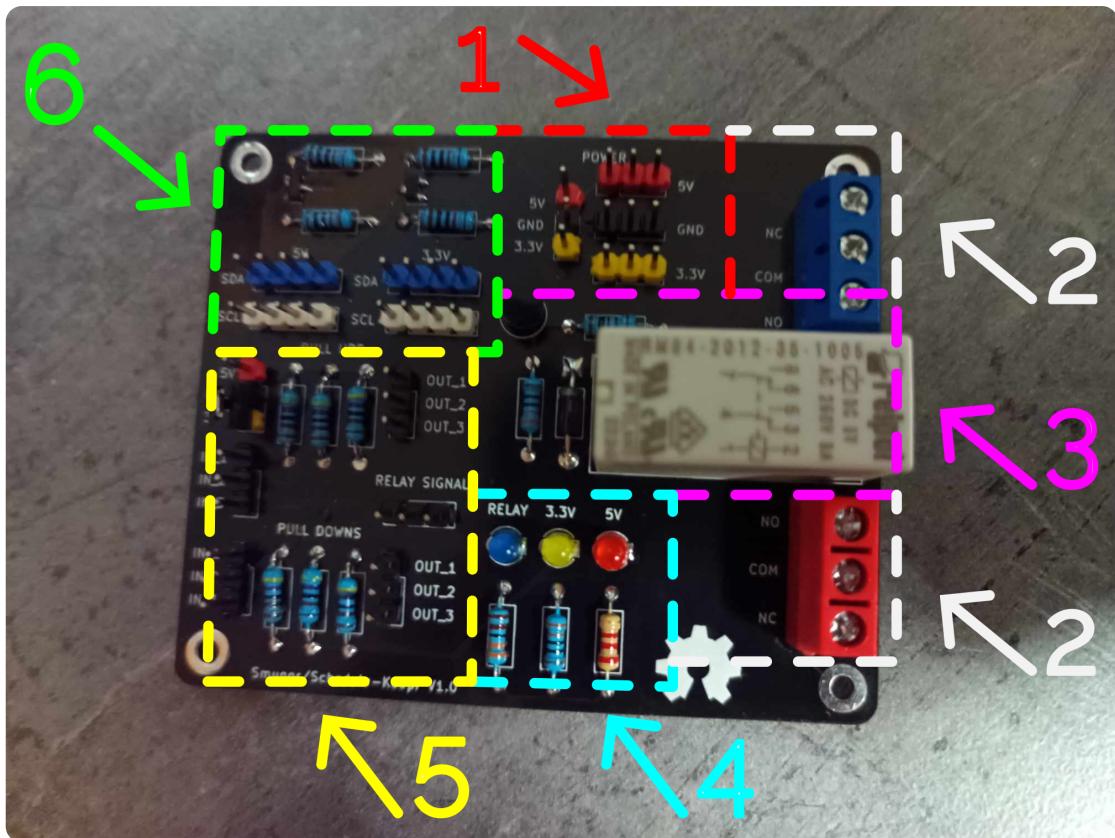
Wizualizacja 3D



Gotowe PCB

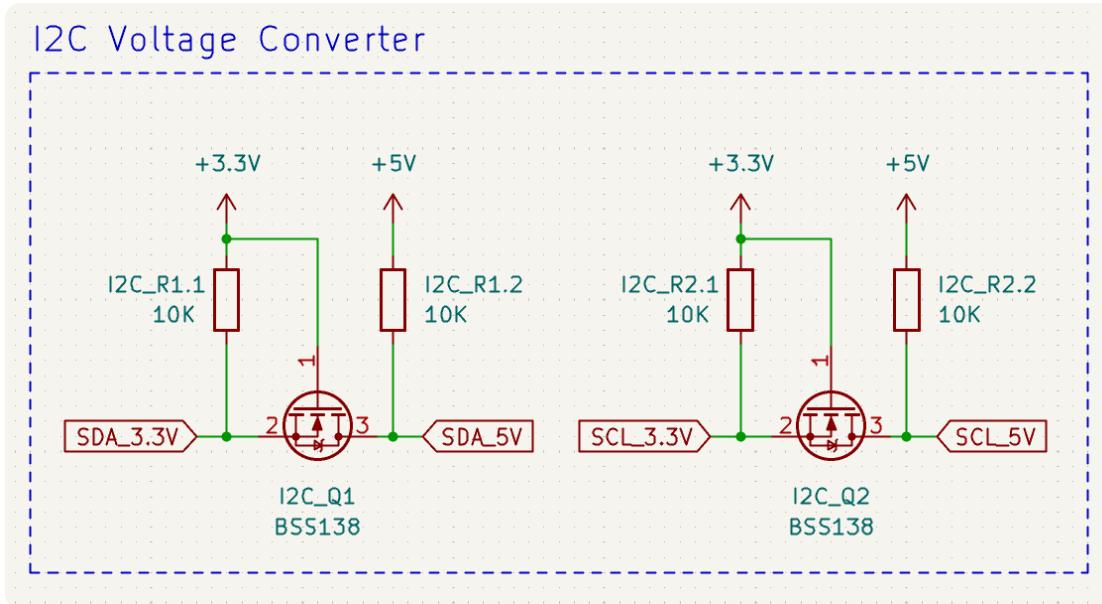


Opis złożenia elementów



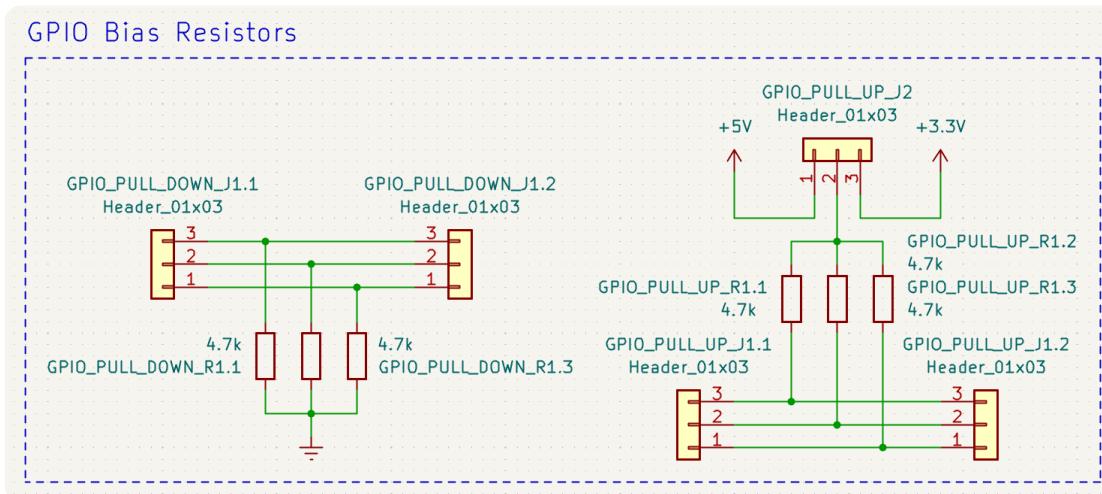
1. Wejście i wyjścia zasilające 5V i 3.3V
2. Złącza śrubowe przekaźnika
3. Przekaźnik wraz z układem sterowania
4. Wskaźniki LED stanu przekaźnika oraz obecności zasilania
5. Wejścia i wyjścia wraz z rezystorami podciągającymi w górę lub w dół
6. Linie I2C o napięciu logicznym 5V oraz 3.3V

Konwerter poziomów logicznych magistrali I2C



Aby zapewnić wszechstronną kompatybilność naszego sterownika z różnorodnymi urządzeniami, zdecydowaliśmy się na wykorzystanie układu (nr. 6) opartego na dwóch tranzystorach **BSS138 (MOSFET typu N)** oraz czterech rezystorach. Ten układ jest przeznaczony do konwersji poziomów logicznych, co umożliwia skutecną integrację naszego sterownika z różnymi urządzeniami wykorzystującymi protokół komunikacyjny I2C.

Rezystory podciągające w góre lub dół



Na płytce znalazło się również wiele przydatnych wyprowadzeń, w tym wyprowadzenia z rezystorami podciągającymi w góre lub w dół (nr. 5). Te rezystory są kluczowe dla umożliwienia podpięcia różnych czujników, urządzeń wejściowych (takich jak przełączniki czy guziki) a nawet GPIO komputera jednopłytkowego znajdującego się w środku. Ich obecność zapewnia nie tylko elastyczność w integracji z różnymi układami lub urządzeniami, ale także stabilność sygnałów logicznych, co gwarantuje niezawodną pracę naszego sterownika w różnorodnych warunkach użytkowania.

Na tej płytce znajdują się również 2 wyjścia śrubowe od przekaźnika (nr. 2), które są używane do przesterowywania podłączonych do nich urządzeń. Dodatkowo, umieszczone zostały wskaźniki LED (nr. 4) - niebieski wskaźnik informuje o stanie cewki przekaźnika, żółty oznacza zasilanie 3.3V, natomiast czerwony

sygnalizuje zasilanie 5V - elementy te nie tylko zapewniają kontrolę nad działaniem urządzenia, ale także umożliwiają szybką diagnostykę stanu pracy sterownika, co przyczynia się do sprawnego monitorowania oraz konserwacji systemu.

5.3. Gotowy produkt - program

Dedykowany program do sterownika został napisany w językach programowania **Go** oraz **Bash**,