

Spis Treści

- [Spis Treści](#)
 - [1. Wstęp](#)
 - [2. Cele i założenia](#)
 - [3. Przebieg pracy](#)
 - [4. Część Informatyczna](#)
 - [4.1 Backend - strona serwerowa](#)
 - [4.2 Frontend - strona użytkownika](#)

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 kompilacji 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 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** wersja 1.9.1
- **COMQTT** wersja 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) pozwoliły na proste wbudowanie dodatkowych zabezpieczeń dostępu do serwera REST, takie 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 COMQTT 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 uwierzytelniania 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.

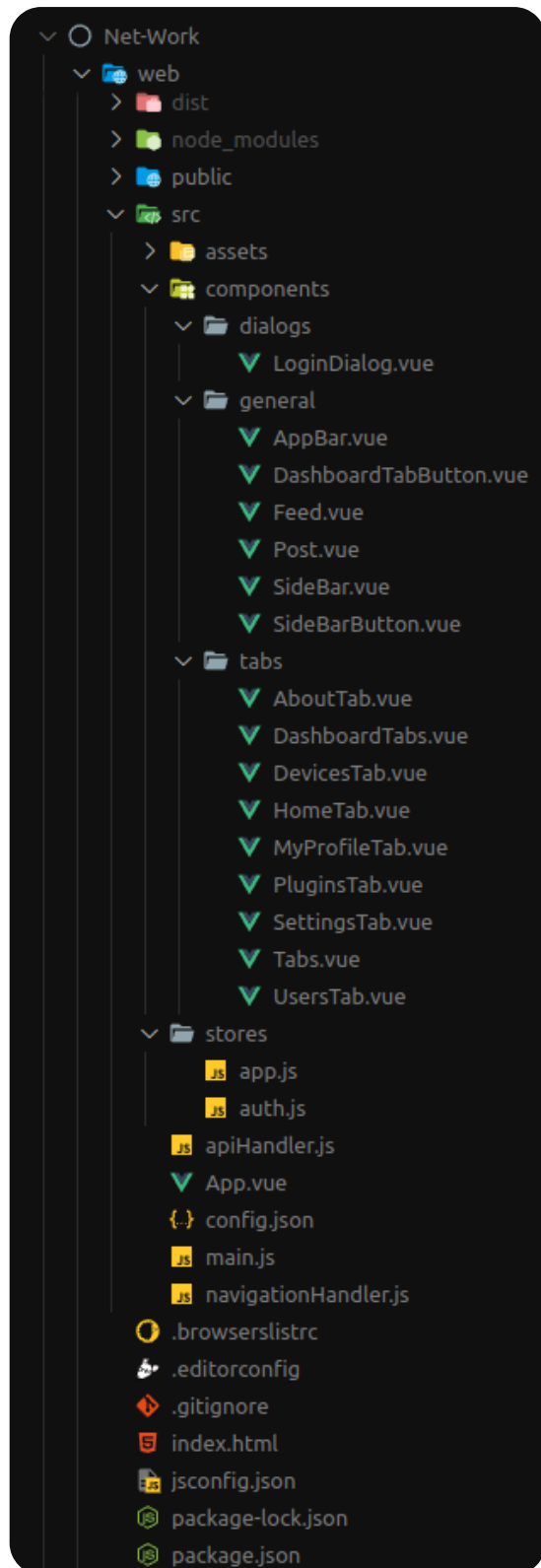
Dzięki wykorzystaniu tych pakietów możliwe było zrealizowanie nie tylko bezpiecznego, ale także wydajnego serwera REST oraz brokera MQTT, który spełnia wymagania zarówno pod kątem funkcjonalności, jak i wydajności.

4.2 Frontend - strona użytkownika

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

Aplikacja internetowa oparta jest na nowoczesnych technologiach takich jak framework **Vue.js w wersji 3**, **Vite w wersji 5.2.2** jako narzędzie do budowania projektu w jedną spójną całość gotową do uruchomienia, oraz Vuetify, biblioteki komponentów **Material Design**. W połączeniu z biblioteką **Pinia w wersji 2.1.7** do zarządzania stanem aplikacji oraz **Axios w wersji 1.6.7** do komunikacji z serwerem, te technologie pozwoliły nam na stworzenie wyjątkowo wydajnej aplikacji jednostronowej **SPA (Single Page Application)**. Vue.js umożliwia dynamiczne routowanie po stronie użytkownika, eliminując jednocześnie konieczność przetadowywania całej strony podczas przejść między 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.

Struktura plików frontendu



Przykładowy komponent

```
1 <template>
2   <v-window-item :value="value" v-model="selectedSubTabName">
3     <v-tabs
4       v-model="selectedSubTabName"
5       align="center"
6       color="deeph-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="g"
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>
```