Spis Treści

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 ***Represantional 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 integracje 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 uwierzetelniania 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.

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* 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 jedna 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 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

```vue
<template>
  <v-window-item :value="value" v-model="selectedSubTabName">
    <v-tabs
      v-model="selectedSubTabName"
      align-tabs="center"
      color="deeep-purple-accent-4"
      mandatory
      stacked
    >
      <slot name="buttons" />
    </v-tabs>

    <v-window
      v-model="selectedSubTabName"
      direction="horizontal"
      elevation="0"
      mandatory
    >
      <slot name="content" />
    </v-window>
  </v-window-item>
</template>

<script>
export default {
  name: 'DashboardTab',
  props: {
    value: {
      type: Object,
      required: true,
    },
    childValue: {
      type: Object,
      required: true,
    },
  },
  computed: {
    selectedSubTabName: {
      get() {
        console.log('getting dashboard tab name ', this.value);
        return this.childValue;
      },
      set(newValue) {
        if (this.childValue && newValue !== this.childValue) {
          console.log('updating dashboard tab name ', newValue);
          this.$emit('update:childValue', newValue );
        }
      },
    },
  },
  mounted() {
    console.log('DashboardTab component received value and childValue props:', this.value, this.childValue);
  }
};
</script>
```

Struktura plików frontendu