

Programowanie Aplikacji Internetowych

Projekt

Skład Projektowy:

Jagoda Ługowska	331505
Mateusz Skorek	331532
Nina Kałuża	331489

Spis treści

1	Zakres i cel projektu	3
2	Zespół	3
3	Założenia funkcjonalności	3
4	Architektura systemu	3
5	Opis realizacji	4
5.1	Komunikacja mikrokontrolera z ThinkSpeak	4
5.2	Struktura bazy danych	5
5.3	Backend – przetwarzanie danych z chmury	5
5.4	Moduł: FeedService	5
5.4.1	Opis	5
5.4.2	Główne metody	6
5.4.3	Zależności	6
5.5	Moduł: FeedController	6
5.5.1	Opis	6
5.5.2	Główne endpointy	6
5.5.3	Zależności	6
5.6	Relacje między modułami	6
5.7	Moduł: ChartController	7
5.7.1	Opis	7
5.7.2	Główne endpointy	7
5.7.3	Parametry	7
5.8	Moduł: UserController	7
5.8.1	Opis	7
5.8.2	Główne endpointy	7
5.9	Moduł: UserService	7
5.9.1	Opis	7
5.9.2	Zastosowane techniki	8
5.9.3	Wybrane metody	8
5.10	Encje: User, UserOTD, UserResponse	8
5.10.1	User	8
5.10.2	UserOTD (Object To Deliver)	8
5.10.3	UserResponse	8
5.11	Frontend – prezentacja danych	8
5.12	Frontend - system logowania	9
5.13	Frontend - panel administratora	9
5.14	Frontend - dashboard	10
5.15	Frontend - panel rejestracji	10
6	Interfejs użytkownika i działanie aplikacji	11
6.1	Widok niezalogowanego użytkownika	11
6.2	Widok zalogowanego użytkownika	12
6.3	Rejestracja	12
6.4	Panel administratora	13
7	Testy	13
8	Podsumowanie i Wnioski	14

1 Zakres i cel projektu

Celem projektu było stowrzenie aplikacji webowej służącej do monitorowania zużycia energii elektrycznej przez różne źródła światła (halogeny i LED-y). Dane są zbierane z fizycznego układu pomiarowego i prezentowane użytkownikowi w czasie rzeczywistym oraz w formie analizy historycznej.

Do realizacji projektu wykorzystaliśmy następujące narzędzia:

- Thonny - środowisko programistyczne z wbudowanym debuggerem, wykorzystywane do programowania mikrokontrolera w języku MicroPython
- Visual Studio Code – rozbudowane środowisko programistyczne wykorzystane do tworzenia frontendu i backendu aplikacji
- ThinkSpeak - platforma IoT służąca do gromadzenia i wizualizacji danych pomiarowych w chmurze
- MongoDB - baza danych NoSQL wykorzystana do przechowywania historycznych pomiarów zużycia energii
- Docker - narzędzie konteneryzacji użyte do spakowania i wdrożenia aplikacji

2 Zespół

- Nina Kałuża - Frontend / Baza Danych / Project Manager
- Mateusz Skorek - Backend / Baza Danych/ DevOps
- Jagoda Ługowska - Frontend / deployment (Docker) / IoT

3 Założenia funkcojalności

Monitoring – podgląd wartości napięcia, natężenia, mocy oraz zużycia energii z danych zbieranych co

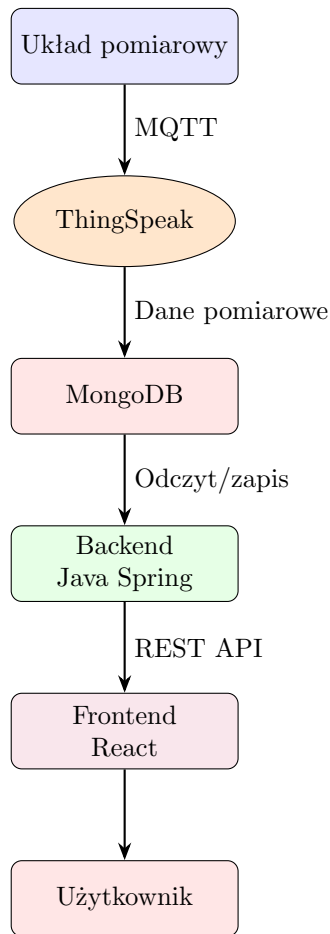
Wizualizacja danych – interaktywne wykresy oraz tabele danych z możliwością filtrowania.

Analiza historyczna – możliwość przeglądania danych archiwalnych (maksymalnie do miesiąca wstecz) i ich eksport (CSV/Excel) tylko dla zalogowanych użytkowników. minutę.

System Logowania - stworzenie własnego systemu logowania/tworzenia kont użytkowników dzięki czemu będą mieli oni większe możliwości działania na stronie.

Panel Adminstratora - stworzenie panelu administratora do między innymi: edytowania/usuwania/dodawania kont użytkowników, edytowania/usuwania/dodawania zebranych danych w razie potrzeby jakichkolwiek poprawek.

4 Architektura systemu



Rys. 1: Architektura systemu monitorowania energii

5 Opis realizacji

5.1 Komunikacja mikrokontrolera z ThingSpeak

W projekcie zastosowano mikrokontroler ESP32 z oprogramowaniem w języku MicroPython, którego zadaniem było przesyłanie danych pomiarowych do chmury, a konkretnie do platformy ThingSpeak. Komunikacja z chmurą odbywa się z wykorzystaniem protokołu MQTT.

Połączenie z siecią Wi-Fi

Mikrokontroler łączy się z lokalną siecią Wi-Fi przy użyciu biblioteki network. Proces połączenia obejmuje aktywację interfejsu sieciowego, podanie danych dostępowych i oczekiwanie na ustanowienie połączenia. Po połączeniu, uzyskiwana jest konfiguracja sieci IP.

Połączenie z ThingSpeak przez MQTT Po stronie ThingSpeak dane są odbierane za pośrednictwem specjalnie skonfigurowanego kanału MQTT. ESP32 łączy się z brokerem `mqtt3.thingspeak.com`, korzystając z unikalnych danych autoryzacyjnych wygenerowanych w ustawieniach kanału:

- `MQTT_CLIENT_ID`, `MQTT_USERNAME`, `MQTT_PASSWORD` – dane identyfikujące i uwierzytelniające klienta MQTT
- `TOPIC` – temat publikacji danych w formacie `channels/channelID/publish`

Publikacja danych pomiarowych

W danych publikowanych przez mikrokontroler znajdują się odczyty napięcia i dwóch niezależnych wartości natężenia prądu. Wartości te przypisywane są odpowiednim polom ThingSpeak (`field1`, `field2`, `field3`). Dane są przesyłane w postaci tekstowego zapytania. Przykład wiadomości przesyłanej do ThingSpeak może wyglądać następująco:

$field1 = 11.72 \& field2 = 0.87 \& field3 = 1.45$

Dane te są automatycznie rejestrowane przez ThingSpeak i wizualizowane na wykresach w panelu użytkownika kanału. **Obsługa błędów**

Całość programu umieszczono w bloku try/except, który pozwala przechwycić i zidentyfikować ewentualne błędy występujące w czasie.

5.2 Struktura bazy danych

W projekcie zastosowano nierelacyjną bazę danych MongoDB, która cechuje się dokumentowym modelem przechowywania danych. Cała baza nosi nazwę **PAINT** i zawiera kilka kolekcji danych, które pełnią różne funkcje w systemie. Struktura bazy została zorganizowana w następujący sposób:

1. Kolekcje `feeds_<data>`

- Są to kolekcje tworzone dynamicznie, gdzie `<data>` odpowiada konkretnej dacie w ustalonym formacie (np. `feeds_2025_06_08`).
- Każda z tych kolekcji zawiera dokumenty w formacie JSON, które reprezentują pojedyncze wpisy użytkowników (tzw. *feedy*).
- Dzięki zastosowaniu takiego schematu możliwe jest logiczne grupowanie danych według dat, co może ułatwiać filtrowanie i zarządzanie zawartością.

2. Kolekcja `users`

- Zawiera dane zarejestrowanych użytkowników.
- Każdy dokument reprezentuje jednego użytkownika i przechowuje informacje w formacie JSON, takie jak login, haszowane hasło, adres e-mail, itp.
- Struktura danych użytkownika jest elastyczna i może być łatwo rozszerzana o kolejne pola (np. ustawienia konta czy preferencje).

3. Kolekcja `counters`

- Odpowiada za generowanie unikalnych identyfikatorów (`id`) dla nowych użytkowników.
- Jest to pomocnicza kolekcja, w której przechowywany jest aktualny stan licznika identyfikatorów.
- Mechanizm ten pozwala na zachowanie ciągłości i unikalności ID, co jest szczególnie przydatne w przypadku braku autoinkrementacji typowej dla relacyjnych baz danych.

Uwagi dotyczące struktury dokumentów

Każdy dokument w kolekcjach jest zapisany w formacie JSON (JavaScript Object Notation), który umożliwia łatwe przechowywanie złożonych struktur danych, w tym zagnieżdżonych obiektów i tablic. Z uwagi na obszerność i zmienność schematów dokumentów (szczególnie w kolekcjach `feeds_< data >`, ich dokładna struktura nie została wyszczególniona w tym sprawozdaniu, jednak opiera się ona na prostych typach danych takich jak tekst, liczby, znaczniki czasu oraz może posiadać różną liczbę pól danych.

5.3 Backend – przetwarzanie danych z chmury

5.4 Moduł: `FeedService`

Pakiet: `com.mires.paint.services.feed`

5.4.1 Opis

Moduł `FeedService` odpowiada za interakcję z bazą danych MongoDB. Dane są zapisywane w kolekcjach nazwanych dynamicznie w zależności od daty (`feeds_yyyy_MM_dd`). Używany jest mechanizm `upsert`, który pozwala na aktualizację lub dodanie nowego dokumentu.

5.4.2 Główne metody

- `saveFeeds(List<Feed> feeds)` – zapisuje listę obiektów `Feed` do MongoDB.
- `upsertFeed(Feed feed)` – zapisuje pojedynczy obiekt `Feed` z wykorzystaniem opcji `upsert`.
- `getFeedFromDbByID(double id, LocalDateTime createdAt)` – pobiera pojedynczy wpis z bazy danych.
- `getFeedsFromDb(LocalDateTime start, LocalDateTime end)` – pobiera dane z zakresu czasowego z wielu kolekcji MongoDB.

5.4.3 Zależności

- `MongoClient` – klient MongoDB.
- `Document` (`org.bson`) – reprezentacja dokumentu w MongoDB.
- `Feed` – encja reprezentująca dane z kanału zewnętrznego (np. `ThingSpeak`).

5.5 Moduł: `FeedController`

Pakiet: `com.mires.paint.controllers.feeds`

5.5.1 Opis

Moduł `FeedController` zapewnia REST API do obsługi danych typu `Feed`. Umożliwia pobieranie danych z zewnętrznego API (`ThingSpeak`), zapis do lokalnej bazy danych oraz zwracanie danych na żądanie klienta frontendowego.

5.5.2 Główne endpointy

- `GET /api/feeds/` – pobiera dane z zewnętrznego API i zapisuje je w MongoDB.
- `GET /api/feeds/range` – zwraca dane z MongoDB z podanego zakresu czasowego.
- `POST /api/feeds/edit` – umożliwia edycję i zapis pojedynczego wpisu.
- `GET /api/feeds/push-edits` – wysyła lokalne edycje do API `ThingSpeak` (bulk update).

5.5.3 Zależności

- `FeedService` – główny serwis odpowiedzialny za operacje na danych.
- `WebClient` – klient HTTP do komunikacji z zewnętrznym API.
- `Gson` – biblioteka do parsowania JSON (z `GsonFactory`).
- `FeedResponse`, `ErrorResponse` – klasy reprezentujące odpowiedzi API.

5.6 Relacje między modułami

- `FeedController` korzysta bezpośrednio z `FeedService` jako warstwy logiki biznesowej.
- `FeedService` izoluje operacje na bazie danych, umożliwiając testowalność i ponowne wykorzystanie logiki.
- Przepływ danych:
 1. Controller żąda danych lub aktualizacji.
 2. Service przetwarza i zapisuje dane w MongoDB.
 3. W przypadku braku danych lokalnych – Controller aktualizuje dane przez wywołanie `GET /api/feeds/`.

5.7 Moduł: ChartController

Pakiet: `com.mires.paint.controllers.charts`

5.7.1 Opis

Moduł `ChartController` udostępnia endpoint RESTowy do pobierania dynamicznych wykresów HTML ze zdalnego API (ThingSpeak). W zależności od przekazanych parametrów (takich jak pole, kolor, rozmiar, zakres czasowy), generuje odpowiedni URI i pobiera gotowy kod HTML.

5.7.2 Główne endpointy

- GET `/api/chart` – zwraca kod HTML wykresu z serwisu ThingSpeak.

5.7.3 Parametry

- `field` – numer pola kanału (domyślnie 1).
- `title` – tytuł wykresu.
- `width`, `height` – rozmiary wykresu.
- `start`, `end` – zakres dat (domyślnie ostatnie 2 godziny).
- `color` – kolor wykresu w formacie hex.

5.8 Moduł: UserController

Pakiet: `com.mires.paint.controllers.user`

5.8.1 Opis

Kontroler zarządza operacjami użytkowników, takimi jak rejestracja, logowanie, edycja, usuwanie i listowanie użytkowników. Korzysta z reaktywnego serwisu `UserService`.

5.8.2 Główne endpointy

- POST `/api/users/create` – tworzy nowego użytkownika.
- POST `/api/users/login` – loguje użytkownika.
- POST `/api/users/edit` – aktualizuje dane użytkownika.
- POST `/api/users/delete` – usuwa użytkownika po ID.
- GET `/api/users/list` – zwraca listę wszystkich użytkowników.

5.9 Moduł: UserService

Pakiet: `com.mires.paint.services.user`

5.9.1 Opis

Logika biznesowa zarządzania użytkownikami, w tym:

- tworzenie unikalnych identyfikatorów,
- walidacja logowania,
- edycja danych,
- usuwanie użytkowników z bazy MongoDB.

5.9.2 Zastosowane techniki

- Upsert z użyciem kolekcji `counters` do inkrementacji ID.
- Obsługa błędów przy pomocy klasy `ErrorResponse`.
- Programowanie reaktywne z wykorzystaniem `Mono` i `Flux`.

5.9.3 Wybrane metody

- `createUser(UserOTD)` – tworzy nowego użytkownika z unikalnym ID.
- `findByLogin(String)` – wyszukuje użytkownika po loginie.
- `findById(int)` – wyszukuje użytkownika po ID.
- `deleteUser(int)` – usuwa użytkownika z bazy danych.
- `updateUser(User)` – aktualizuje dane użytkownika.
- `login(String, String)` – sprawdza poprawność danych logowania.
- `listUsers()` – zwraca listę wszystkich użytkowników.

5.10 Encje: User, UserOTD, UserResponse

5.10.1 User

- Klasa reprezentująca użytkownika w bazie danych.
- Annotacja `@BsonId` identyfikuje pole `_id` jako klucz główny.
- Pola: `_id`, `login`, `password`, `email`, `name`, `surname`, `role`.

5.10.2 UserOTD (Object To Deliver)

- DTO do odbioru danych rejestracyjnych z klienta.
- Nie zawiera pola `_id`.

5.10.3 UserResponse

- Wrapper odpowiedzi zwracanej klientowi.
- Pola: `user` – obiekt użytkownika (może być null), `errorResponse` – opis błędu.

5.11 Frontend – prezentacja danych

Frontend aplikacji został zaprojektowany w sposób responsywny i przejrzysty, z naciskiem na czytelność prezentowanych danych pomiarowych. Interfejs użytkownika został przygotowany przy użyciu HTML, CSS oraz JavaScript.

Główne funkcje interfejsu:

- Strona główna – zawiera ogólną prezentację systemu, opcję logowania oraz dynamiczne wykresy wyświetlające dane (dla niezalogowanych użytkowników).
- Panel użytkownika (po zalogowaniu) – umożliwia dostęp do: analizy danych historycznych, eksportowania danych do pliku CSV
- Panel administratora – dostępny wyłącznie dla użytkowników z odpowiednimi uprawnieniami. Umożliwia zarządzanie kontami użytkowników (dodawanie, usuwanie, edytowanie) oraz poprawianie/edycję danych pomiarowych w razie błędów transmisji.

5.12 Frontend - system logowania

System logowania został zaimplementowany jako komponent Reacta o nazwie LoginForm, który odpowiada za uwierzytelnienie użytkownika oraz przekierowanie go do odpowiedniej części aplikacji w zależności od przypisanej roli.

Funkcjonalność komponentu:

- Formularz zawiera dwa pola: login i hasło
- Po zatwierdzeniu formularza wywoływana jest funkcja login z kontekstu UserContext, która przesyła dane do backendu w celu weryfikacji
- Jeśli logowanie zakończy się sukcesem:
 - użytkownik zostaje przekierowany na /admin, jeśli jego rola to "admin"
 - w przeciwnym razie na /dashboard.
- W przypadku błędnych danych logowania, użytkownik otrzymuje komunikat: „Nieprawidłowy login lub hasło.”

Nawigacja

- Dodatkowo użytkownik ma dostęp do przycisków:
 - „Zarejestruj się” – prowadzi do formularza rejestracji (/register)
 - „Nie pamiętasz hasła?” – przekierowuje do strony resetowania hasła (/reset-password)

Stylizacja

Formularz jest stylizowany za pomocą klasy LoginForm.css, co umożliwia responsywny i estetyczny wygląd interfejsu logowania.

Bezpieczeństwo i UX

Pola są oznaczone jako required, dzięki czemu przeglądarka blokuje wysłanie formularza bez podania danych. Hasło jest maskowane (type="password"), co chroni przed nieautoryzowanym podglądem.

5.13 Frontend - panel administratora

AdminPanel to główny komponent panelu administracyjnego aplikacji, umożliwiający zarządzanie użytkownikami, przeglądanie i filtrowanie danych pomiarowych, wyświetlanie wykresów oraz edycję i eksport danych.

Panel jest podzielony na trzy główne zakładki:

- Użytkownicy — zarządzanie użytkownikami (edycja, usuwanie)
- Wykresy — wizualizacja danych pomiarowych w formie wykresów
- Dane — edycja i eksport danych numerycznych

Funkcje

- Zarządzanie użytkownikami: edytuj, usuń
- Filtrowanie danych po dacie
- Eksport danych do CSV
- Edycja danych w tabeli

Komponenty pomocnicze

- FilteringBox — formularz filtrowania dat i eksportu
- Modal — modale do zmiany hasła i edycji użytkownika
- AccordionChart — wykresy danych
- EditableDataTable — edytowalna tabela danych

5.14 Frontend - dashboard

Komponent Dashboard po zalogowaniu to główny ekran użytkownika po zalogowaniu.

Struktura danych

- **Kontekst użytkownika:** wykorzystuje `useUser()` do pobrania danych aktualnie zalogowanego użytkownika (np. email)
- **Stan komponentu:**
 - `loading` – określa czy trwa pobieranie danych
 - `error` – przechowuje komunikaty o błędach
 - `chartData` – dane przetworzone do wyświetlenia na wykresach
 - `startDate` i `endDate` – zakres czasowy do filtrowania danych

Proces działania

1. Po załadowaniu komponentu (`useEffect`) następuje automatyczne pobranie danych w domyślnym zakresie (`start` i `end`)
2. Funkcja `loadData`:
 - wywołuje API `api.getFeedsInRange(start, end)`
 - zabezpiecza sytuację, gdy zwrócone dane są `null`
 - transformuje surowe dane przez `transformApiData`
 - tworzy strukturę danych do wykresów
 - ustawia dane i zarządza stanem błędu/loading
3. Formularz umożliwia ręczne ustawienie zakresu dat i odświeżenie danych
4. W przypadku błędu podczas pobierania, wyświetla się komunikat
5. Trzy wykresy są wyświetlane w formie `iframe` z linkami generowanymi dynamicznie przez `api.getChartUrl`

Interfejs użytkownika

- Nagłówek z powitaniem i emailem użytkownika
- Sekcja formularza wyboru zakresu dat
- Sekcja wykresów z:
 - wskaźnikiem ładowania
 - obsługą błędów

5.15 Frontend - panel rejestracji

Komponent `RegisterForm` odpowiada za rejestrację nowych użytkowników w aplikacji. Zawiera formularz umożliwiający podanie danych osobowych oraz weryfikację poprawności haseł i adresu e-mail. Po pomyślnym zakończeniu procesu użytkownik zostaje przekierowany do strony logowania.

Formularz rejestracyjny umożliwia wprowadzenie następujących danych:

- Imię (`name`)
- Nazwisko (`surname`)
- Adres e-mail (`email`)
- Hasło (`password`)

- Potwierdzenie hasła (confirmPassword)

Walidacja danych:

- Porównanie haseł – system sprawdza, czy hasło i jego potwierdzenie są identyczne
- Format e-maila – zastosowano wyrażenie regularne do walidacji zgodności adresu z ogólnymi zasadami (np. brak spacji, obecność @ i domeny)

Rejestracja użytkownika

- Funkcja register pochodzi z kontekstu UserContext i odpowiada za logikę rejestracyjną
- Po udanej rejestracji użytkownik zostaje automatycznie przekierowany do strony logowania (/login)

Obsługa błędów

- Komunikaty o błędach (np. niezgodność haseł lub niepoprawny e-mail) są wyświetlane bezpośrednio nad formularzem

Nawigacja

- Przycisk „← Wróć” umożliwia szybki powrót na stronę główną
- Pod formularzem znajduje się opcja przekierowania do panelu logowania dla już zarejestrowanych użytkowników

Technologie i zależności: React 18+

React Router (useNavigate) – do nawigacji między widokami

Context API (useUser) – logika zarządzania użytkownikiem (rejestracja)

CSS Modules (LoginForm.css) – stylowanie komponentu

Struktura stanu lokalnego: name, surname, email, password, confirmPassword – dane formularza

error – tekst komunikatu błędu


navigate – hook do nawigacji po zakończeniu rejestracji

6 Interfejs użytkownika i działanie aplikacji


6.1 Widok niezalogowanego użytkownika

Logowanie

Login:



Hasło:

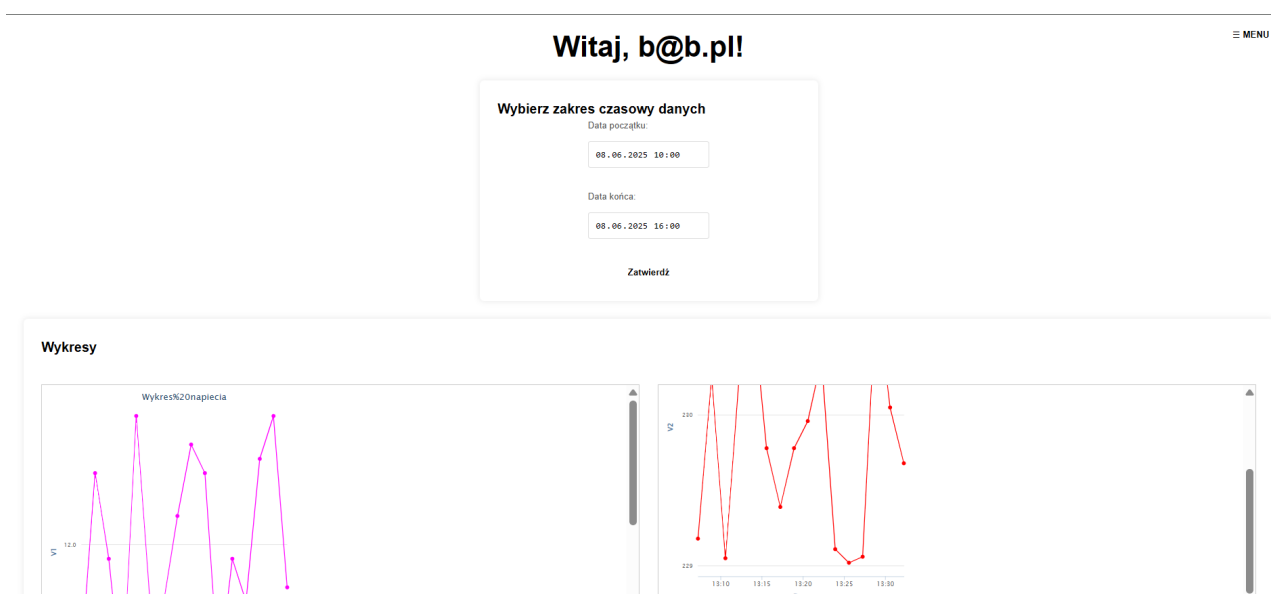


Zaloguj się

Zarejestruj się

Rys. 2: Widok niezalogowanego użytkownika

6.2 Widok zalogowanego użytkownika



Rys. 3: Widok zalogowanego użytkownika

6.3 Rejestracja

Rejestracja

← Wróć

Login:

Name:

Surname:

Email:

Hasło:

Potwierdź hasło:

Zarejestruj się

Masz już konto? Zaloguj się

Rys. 4: Wido krejestracji

6.4 Panel administratora

7 Testy

W ramach realizacji projektu nie zostały zaimplementowane zautomatyzowane testy jednostkowe ani integracyjne. Mimo to przeprowadzono ręczne testy funkcjonalne, które miały na celu sprawdzenie poprawności działania kluczowych funkcji aplikacji z perspektywy użytkownika końcowego.

Panel rejestracji:

- Weryfikacja poprawności walidacji danych (np. niezgodne hasła, nieprawidłowy e-mail)
- Sprawdzenie przekierowania użytkownika do strony logowania po pomyślnej rejestracji
- Obsługa błędów (wyświetlanie komunikatów w przypadku nieprawidłowych danych).

Panel logowania:

- Test logowania z poprawnymi i błędnymi danymi
- Obsługa przypadków braku tokenu lub nieautoryzowanego użytkownika

Dashboard (panel główny):

- Sprawdzenie poprawnego wyświetlania danych (napiecie, prąd, moc, energia)
- Test działania filtrowania danych po zakresie dat
- Responsywność interfejsu oraz poprawność układu graficznego w różnych rozdzielczościach

Ogólna nawigacja i UX:

- Poprawne przekierowania między widokami (/login, /register, /dashboard)
- Widoczność i działanie przycisków (np. „Wróć”, „Zaloguj się”, „Zarejestruj się”)

Testy funkcjonalne pozwoliły na wychwycenie i naprawienie błędów w walidacji formularzy oraz drobnych niedoskonałości w nawigacji. Aplikacja działa stabilnie w zakresie przewidzianych funkcji, a interfejs reaguje poprawnie na działania użytkownika.

8 Podsumowanie i Wnioski

Projekt przedstawia kompletny system do monitorowania parametrów elektrycznych, oparty na architekturze klient-serwer, z wykorzystaniem technologii webowych oraz mikrokontrolera. Aplikacja została wyposażona w funkcjonalny frontend (panel użytkownika, dashboard z wykresami, formularze rejestracji/logowania), backend z obsługą API, a także zrealizowano możliwość konteneryzacji za pomocą Dockera.

W trakcie realizacji projektu udało się z powodzeniem:

- zaimplementować komunikację mqtt
- zaimplementować rejestrację oraz logowanie użytkowników z walidacją danych
- zaprojektować czytelny dashboard prezentujący dane pomiarowe w postaci wykresów
- zastosować dynamiczne filtrowanie danych po zakresie dat
- przetestować ręcznie działanie kluczowych funkcji frontendowych i nawigacyjnych
- przygotować środowisko do uruchamiania aplikacji w kontenerze Docker