



POLITECHNIKA ŚLĄSKA
WYDZIAŁ MATEMATYKI STOSOWANEJ
KIERUNEK INFORMATYKA

Dokumentacja projektu

System zarządzania obowiązkami domowymi działający w czasie
rzeczywistym

Kamil Król,
Mateusz Ostalecki

Gliwice, styczeń 2020

Spis treści

1	Wprowadzenie	1
1.1	Ogólny opis	1
1.2	Linki do aplikacji	1
2	Funkcjonalności aplikacji	2
2.1	Logowanie	2
2.2	Rejestracja	3
2.3	Panel Główny	4
2.3.1	Przegląd i manipulacja pozycjami zadań	4
2.3.2	Nowe zadanie	4
2.3.3	Edycja i podgląd opisu zadania	5
2.3.4	Brak połączenia lub nieprzewidziana akcja	5
3	Wykorzystane technologie	6
3.1	Informacje ogólne	6
3.2	Serwer	6
3.3	Klient	7
4	API servera	7
4.1	Opis	7
4.2	HTTP	7
4.2.1	Rejestracja	8
4.2.2	Logowanie	9
4.3	WebSocket	9
4.3.1	ping	10
4.3.2	task_add	11
4.3.3	task_all	12
4.3.4	task_move	13
4.3.5	task_delete	14
4.3.6	task_edit	15

1 Wprowadzenie

1.1 Ogólny opis

Aplikacja jest systemem zarządzania obowiązkami domowymi w formie witryny internetowej.

Składa się ona z trzech głównych funkcjonalności:

- Strony logowania
- Strony rejestracji
- Strony panelu obsługi obowiązków

Strona logowania jest standardowym systemem autoryzacji dostępu do głównego panelu.

W celu zalogowania należy utworzyć konto w panelu rejestracji.

Główny panel aplikacji posiada w sobie wszystkie funkcjonalności pozwalające na zarządzanie obowiązkami.

1.2 Linki do aplikacji

Działającą aplikację można przetestować pod adresem:

<http://housework-client.herokuapp.com>

Adres serwera znajduje się pod adresem:

<http://housework-api.herokuapp.com/>

Kod źródłowy można znaleźć pod adresem:

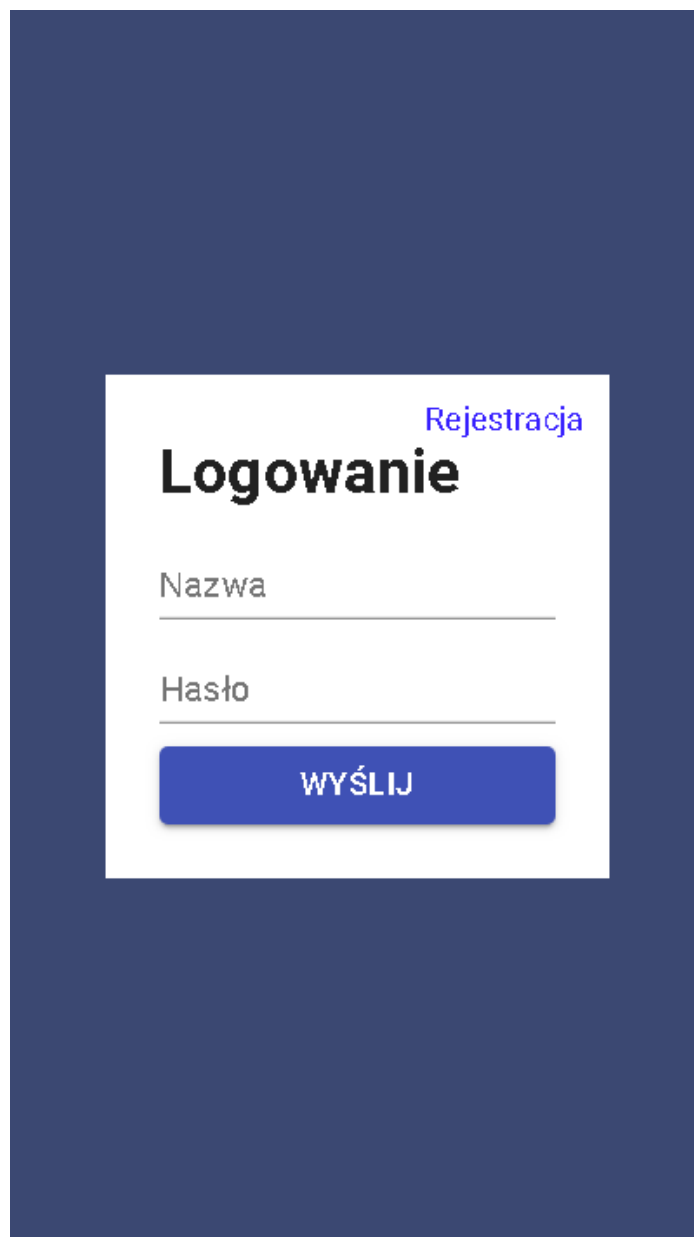
<https://github.com/KrolKamil/housework>

2 Funkcjonalności aplikacji

2.1 Logowanie

Logowanie wymaga podania nazwy użytkownika oraz hasła w celu autoryzacji. Błędne wpisane hasło będzie skutkować podkreśleniem „inputu” na czerwono.

Poprawnie zalogowany użytkownik zostanie przekierowany do głównego panelu.



The image shows a login form centered on a dark blue background. The form is a white rectangle containing the following elements: a link labeled 'Rejestracja' in blue text at the top right; a main title 'Logowanie' in large, bold black text; an input field labeled 'Nazwa' with a thin grey underline; another input field labeled 'Hasło' with a thin grey underline; and a blue button with the text 'WYŚLIJ' in white, bold, uppercase letters.

2.2 Rejestracja

Rejestracja wymaga podania nazwy użytkownika oraz hasła w celu rejestracji. Błędne wpisane hasło będzie skutkowało podkreśleniem „inputu” na czerwono.

Poprawnie zarejestrowany użytkownik zostanie przekierowany do głównego panelu.

The image shows a registration form centered on a dark blue background. The form is a white rectangle containing the following elements: a link 'Logowanie' in blue text at the top right; the title 'Rejestracja' in large, bold black text; a label 'Nazwa' above a text input field; a label 'Hasło' above another text input field; and a blue button with the white text 'WYŚLIJ' at the bottom.

2.3 Panel Główny

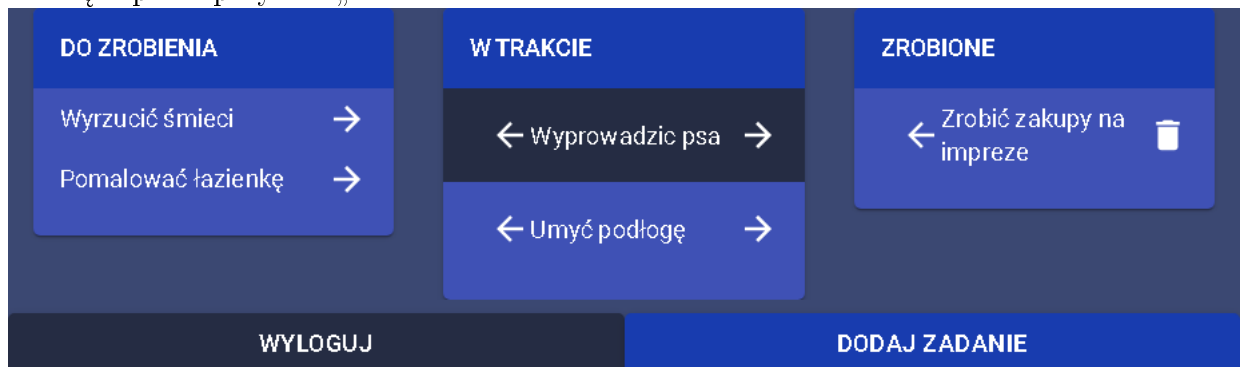
Panel główny posiada najważniejsze funkcjonalności aplikacji:

2.3.1 Przegląd i manipulacja pozycjami zadań

W głównym widoku widzimy wszystkie zdefiniowane zadania.

Jasnoniebieski kolor posiadają zadania które możemy swobodnie przesować pomiędzy kolumnami - są to zadania które jeszcze nikt nie rozpoczął lub czekają w kolumnie „DO ZROBIENIA”.

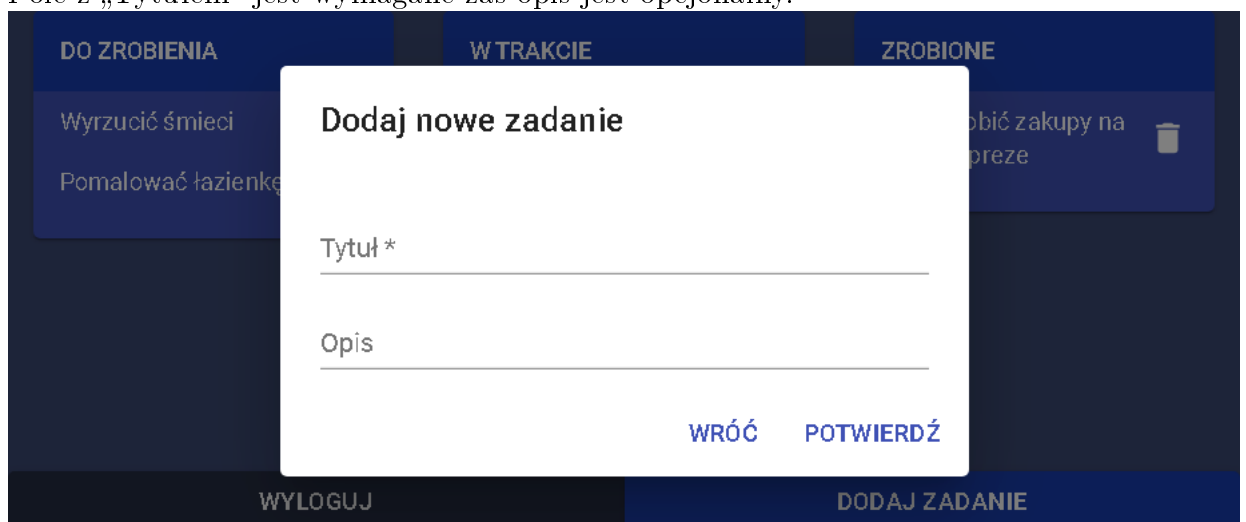
Zadania w ostatniej kolumnie jeśli zostały wykonane przez nas mogą zostać usunięte przez przycisk „kosza”.



2.3.2 Nowe zadanie

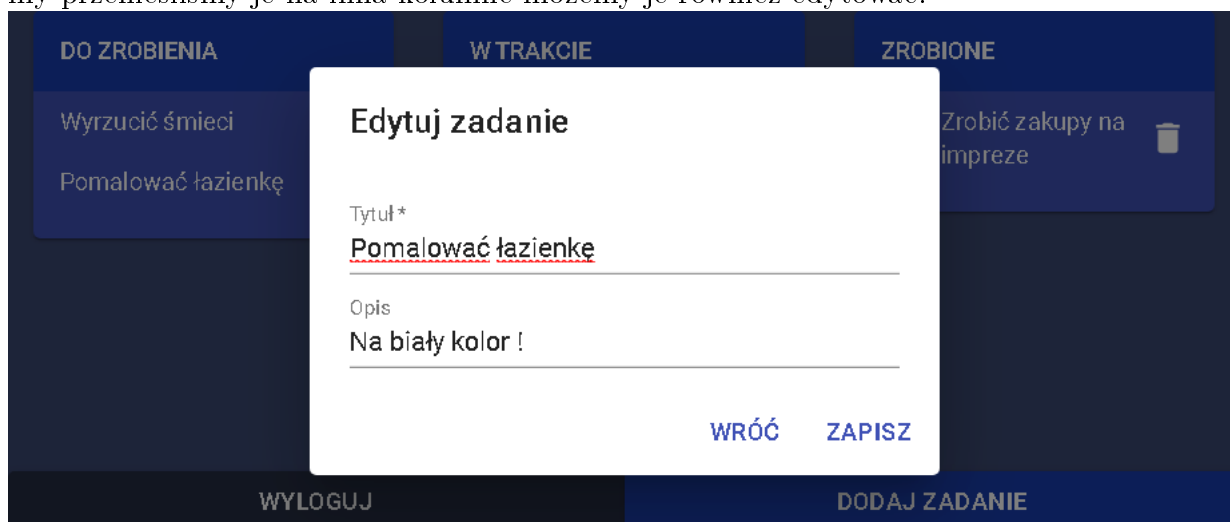
Po kliknięciu w przycisk „DODAJ ZADANIE” aplikacja przeniesie nas do okna dodawania nowego obowiązku.

Pole z „Tytułem” jest wymagane zaś opis jest opcjonalny.



2.3.3 Edycja i podgląd opisu zadania

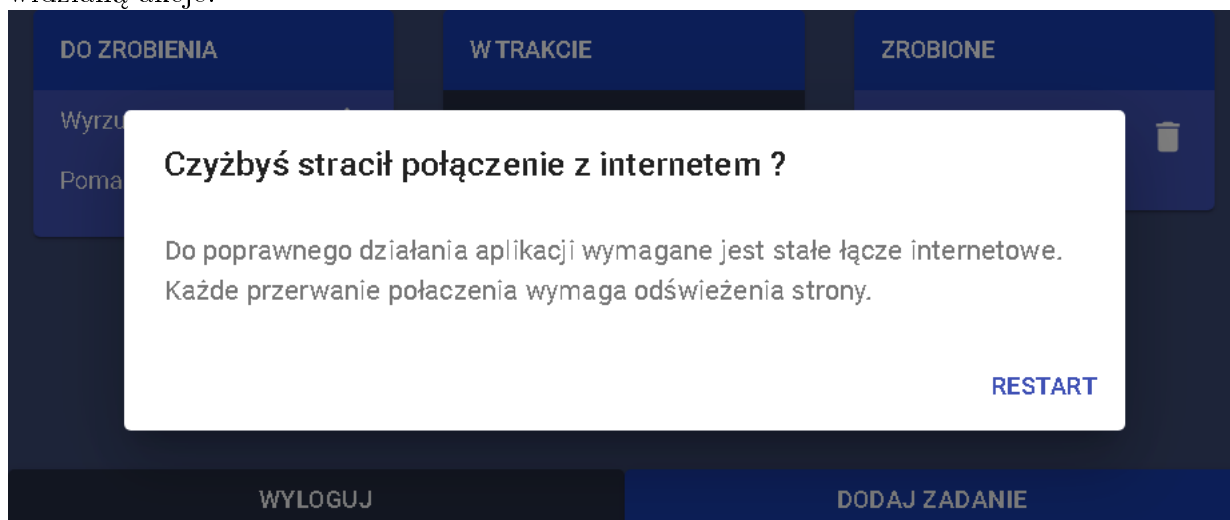
Po kliknięciu w zadanie wyświetli nam się okno w który możemy podejrzeć opis zadania i jeśli zadanie znajduje się w kolumnie „DO ZROBIENIA” lub my przenieśliśmy je na inną kolumnę możemy je również edytować.



2.3.4 Brak połączenia lub nieprzewidziana akcja

Podczas utraty połączenia z internetem użytkownik zostanie poproszony o restart aplikacji.

Ten sam komunikat zostanie wyświetlony jeśli użytkownik wykona nieprzewidzianą akcję.



3 Wykorzystane technologie

3.1 Informacje ogólne

Projekt został w większości wykonany przy użyciu języka JavaScript z uwagi na możliwość wykorzystania go po stronie klienta jak i serwera.

Ciekawym aspektem projektu jest wykorzystanie technologii WebSocket co pozwoliło na aktualizacje danych głównego panelu w czasie rzeczywistym.

Projekt został podzielony na dwa oddzielne podprojekty zwane „mikroserwisami”.

Aplikację klienta jako jeden serwis posiadającą logikę tworzącą interfejs graficzny.

Aplikację serwerową pozwalającą na komunikację z interfejsem graficznym jak i logikę przechowywania i przetwarzania danych.

3.2 Serwer

Serwer wykorzystuje następujące technologie:

- node - platforma uruchomieniowa
- express - serwer http
- ws - serwer WebSocket
- mongodb - baza danych typu noSQL
- JSON Web Token - technologia tworzenia tokenów dostępu

Node pozwala na uruchomienie kodu JavaScript.

Serwer http zapewnia funkcjonalności logowania i rejestracji.

Serwer WebSocket zapewnia wszystkie funkcjonalności związane z zarządzaniem „zadaniami domowymi”.

Baza danych przechowuje informacje dotyczące zarejestrowanych użytkowników oraz „zadań domowych”.

JSON Web Token posiada w sobie logikę do tworzenia tokenu pozwalającego na autoryzację akcji użytkowników.

3.3 Klient

- React - framework do tworzenia interfejsów graficznych
- React-Redux - biblioteka ułatwiająca trzymanie stan aplikacji
- Material UI - biblioteka z gotowymi komponentami
- Axios - biblioteka do XMLHttpRequests

Dzięki „React” i „React-Redux” aplikacja została napisana w sposób ustrukturyzowany podzielony na komponenty przez co zwiększyła się czytelność kodu.

Material UI dostarczył wiele gotowych komponentów graficznych zaskutkowało to zwiększeniem tempa pracy jak i samej estetyki.

Axios znacznie ułatwił prace z API serwera przez opakowanie zapytań w „Promise”.

4 API servera

4.1 Opis

Serwer posiada wiele możliwości komunikacji.

Część autoryzacji wykorzystującą protokół http oraz logikę do przetwarzania „zadań domowych” wykorzystującą protokół WebSocket.

Serwer działa niezależnie od klienta przez co możliwe jest napisanie kompletnie innego interfejsu graficznego.

Opisane w tym rozdziale API dostarczy wszelkich niezbędnych informacji do prawidłowego odpytywania serwera.

4.2 HTTP

Główny adres `http://housework-api.herokuapp.com/` będę oznaczał jako root.

Wszystkie zapytania powinny być typu JSON.

Zatem nagłówki zapytań muszą posiadać: `'Content-Type': 'application/json'`

4.2.1 Rejestracja

POST root/user/register

Zapytanie:

```
{  
  name: „nazwa użytkownika”,  
  password: „hasło użytkownika”  
}
```

Odpowiedź:

Poprawna:

```
{  
  auth: true,  
  token: „token potrzebny do autoryzacji”  
}
```

Błąd:

```
{  
  auth: false,  
  message: „przyczyna błędu”  
}
```

4.2.2 Logowanie

POST root/user/login

Zapytanie:

```
{
  name: „nazwa użytkownika”,
  password: „hasło użytkownika”
}
```

Odpowiedź:

Poprawna:

```
{
  auth: true,
  token: „token potrzebny do autoryzacji”
}
```

Błąd:

```
{
  auth: false,
  message: „przyczyna błędu”
}
```

4.3 WebSocket

Wszystkie zapytania powinny być typu JSON.

Jeżeli dane zapytanie jest błędne serwer w formie odpowiedzi zamknie połączenie.

Lista wszystkich typów zapytań:

- ping
- task_add
- task_all
- task_move
- task_delete
- task_edit

4.3.1 ping

Protokół WebSocket wymaga systemu pingowania w celu utrzymania połączenia.

Następujące zapytanie należy wysłać do serwera w nie większym odstępie niż 30 sekund aby połączenie mogło pozostać utrzymane.

Zapytanie:

```
{  
  type: „ping”  
}
```

Odpowiedź:

```
{  
  type: „pong”  
}
```

4.3.2 task_add

Pozwala na dodanie nowego zadania do bazy

Zapytanie:

```
{
  type: „task_add”,
  payload: {
    token: „token”,
    title: „tytuł zadania”,
    description: „opis zadania”,
    timestamp: „timestamp typu JavaScript”
  }
}
```

Odpowiedź:

```
{
  type: „task_add-confirmation”,
  payload: {
    task: „obiekt typu task”
  }
}
```

Broadcast:

```
{
  type: „task_add”,
  payload: {
    task: „taskObiekt”
  }
}
```

4.3.3 task_all

Pozwala na otrzymanie wszystkich zadan znajdujacych sie w bazie

Zapytanie:

```
{
  type: „task_all”,
  payload:{
    token: „token”
  }
}
```

Odpowiedź:

```
{
  type: „task_all”,
  payload: [taskObiekt, taskObiekt...]
}
```

4.3.4 task_move

Pozwala na zmienienie pozycji tasku

Zapytanie:

```
{
  type: „task_move”,
  payload:{
    token: „token”,
    id: „taskId”,
    position: „TODO” lub „INPROGRESS” lub „DONE”
  }
}
```

Odpowiedź:

```
{
  type: „task_move-confirmation”,
  payload: {
    task: „taskObject”
  }
}
```

Broadcast:

```
{
  type: „task_move”,
  payload: {
    task: „taskObject”
  }
}
```

4.3.5 task_delete

Pozwala na usunięcie zadania

Zapytanie:

```
{
  type: „task_delete”,
  payload: {
    token: token,
    id: „TASK ID”,
  }
}
```

Odpowiedź:

```
{
  type: „task_delete-confirmation”,
  payload: {
    task: „taskObject”
  }
}
```

Broadcast:

```
{
  type: „task_delete”,
  payload: {
    task: „taskObject”
  }
}
```


4.3.6 task_edit

Pozwala na edycje zadania

Zapytanie:

```
{
  type: „task_edit”,
  payload: {
    token: „token”,
    id: „taskId”,
    title: „tytuł”,
    description: „opis”
  }
}
```

Odpowiedź:

```
{
  type: „task_edit-confirmation”,
  payload: {
    task: ”taskObject”
  }
}
```

Broadcast:

```
{
  type: „task_edit”,
  payload: {
    task: ”taskObject”
  }
}
```