



Mercury

Karol Krawczykiewicz, Grzegorz Rogoziński, Jan Król, Piotr Maszczak

Październik 2023 - Styczeń 2024

1 Informacje ogólne

Nazwa projektu: Mercury

Technologie: Express, MongoDB, Neo4j, React, Redux, Sockets, Tailwind, SCSS, TypeScript, WebRTC

Protokoły: HTTP, UDP, ICE, SDP

Repozytorium GitHub: <https://github.com/Karol-2/Mercury-Project>

2 Opis projektu

Mercury jest aplikacją webową, która zapewnia komunikację tekstową oraz na żywo. Aplikacja jest dostosowana do przeglądarek internetowych na komputerach i telefonach, umożliwiając użytkownikom współpracę na różnych urządzeniach. Ponadto, system zapewnia możliwości wyszukiwania, dodawania i usuwania znajomych.

3 Opis działania

3.1 Połączenia

Aplikacja React łączy się z API backendu. Ten zaś, posiada zabezpieczone połączenie z 2 bazami danych. System składa się z 4 elementów:

1. Frontend - działa na `http://localhost:5173`
2. Backend - działa na `http://localhost:5000`
3. Baza Neo4j - przechowuje dane użytkowników, działa na `bolt://localhost:7687` do komunikacji z bazą danych i `http://localhost:7474` do testowania kwerend i wizualizacji danych
4. Baza MongoDB - przechowuje dane o chatach, działa na porcie 27017

API Backendu udostępnia ścieżki dotyczące autoryzacji, obsługi czatów, edycji relacji i obsługi użytkowników. Zapytania są zabezpieczone tokenami.

3.2 Uruchomienie

Aplikację można uruchomić na dwa sposoby:

1. Uruchamiając backend i frontend lokalnie, używając narzędzia npm, i łącząc z lokalnymi wersjami baz danych.
2. Używając kontenerów Docker, dzięki plikowi **compose.yml** z konfiguracją.

4 Modele baz danych

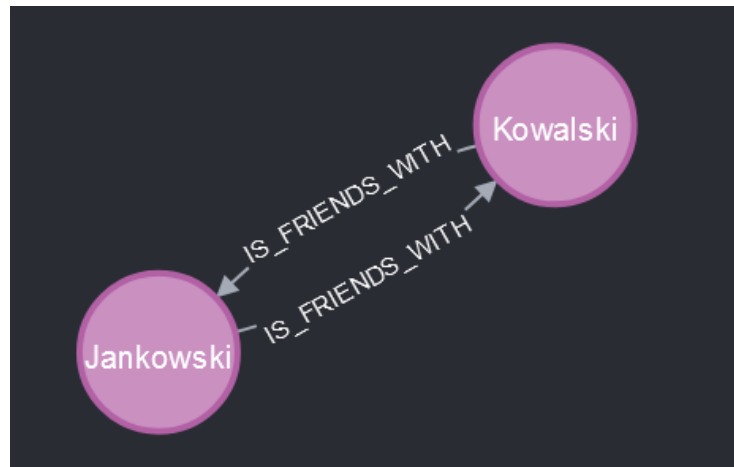
4.1 Model użytkownika

[illegible]

Model użytkownika składa się z siedmiu pól:

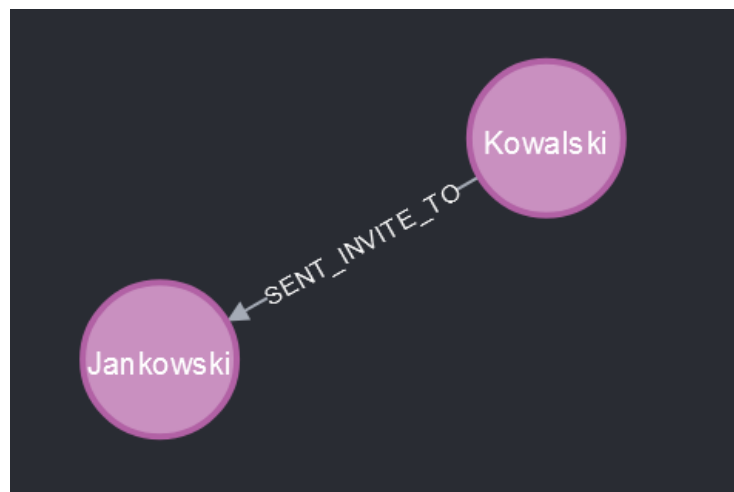
1. **id** - identyfikator
2. **first_name** - imię
3. **last_name** - nazwisko
4. **country** - dwuliterowy kod państwa
5. **profile_picture** - obraz zaszyfrowany base64
6. **mail** - adres email
7. **password** - zaszyfrowane hasło

4.2 Relacje między użytkownikami



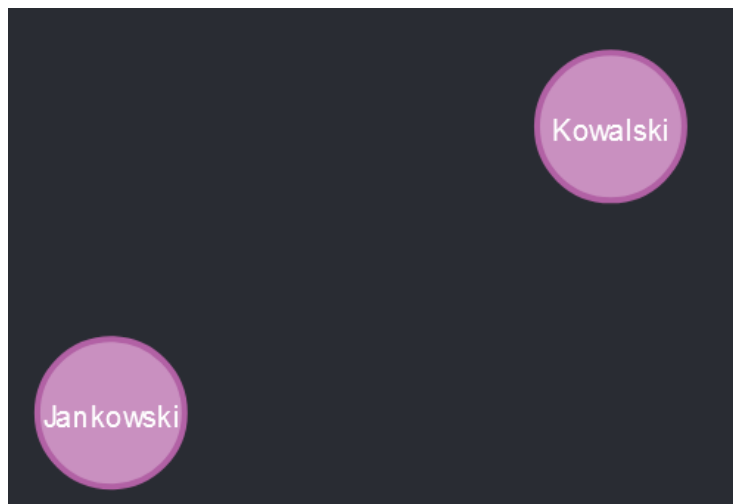
Użytkownicy z dwustronną relacją

Jeżeli dwaj użytkownicy są przyjaciółmi, istnieje między nimi relacja **IS_FRIENDS_WITH**. Obowiązuje ona w dwie strony.



Użytkownicy z jednostronną relacją

W sytuacji gdy jeden użytkownik wyśle zaproszenie do drugiego, nawiązuje się między nimi relacja **SENT_INVITE_TO**. Osoba, od której wychodzi strzałka, wysłała prośbę o dodanie do osoby, przy której jest grot strzałki. Gdy zostaje ono zaakceptowane, usuwane są dotychczasowe relacje i nadawana jest dwustronna relacja **IS_FRIENDS_WITH**.



Użytkownicy bez relacji

W przypadku anulowania prośby lub usunięcia ze znajomych, wszystkie relacje między dwoma użytkownikami zostają usunięte.

4.3 Model chatu

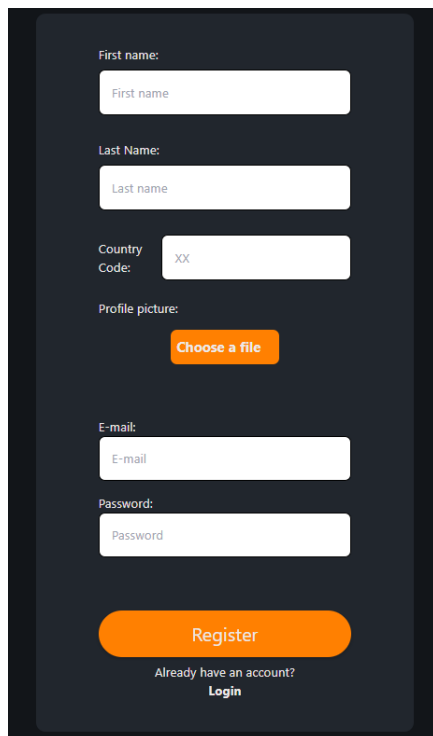
```
{
  _id: ObjectId('65a178a4800d927bf5b414d5'),
  authorId: '6e444ac6-4cea-4462-9134-7301445a6e28',
  receiverId: '5d16d302-ca25-498a-9865-bb9b867234b2',
  content: 'Hello World!',
  created_date: ISODate('2024-01-12T17:36:36.085Z'),
  __v: 0
}
```

Model chatu zawiera dane o wiadomościach tekstowych przesyłanych między użytkownikami. Składają się na niego pola: **id** (id wiadomości), **authorId** (id nadawcy), **receiverId** (id odbiorcy), **content** (wiadomość), **created_date** (data wysłania).

5 Najważniejsze systemy

5.1 System rejestracji i logowania

Rejestracja nowego użytkownika odbywa się po wybraniu opcji Register na ekranie głównym, lub poprzez wejście na odpowiedni endpoint:
`http://localhost:5173/register`

A registration form with a dark background. It contains several input fields: 'First name' and 'Last Name' (both with placeholder text 'First name' and 'Last name' respectively), 'Country Code' (with placeholder 'XX'), 'E-mail' (with placeholder 'E-mail'), and 'Password' (with placeholder 'Password'). There is an orange button labeled 'Choose a file' for the profile picture. At the bottom, there is a large orange 'Register' button and a link 'Already have an account? Login'.

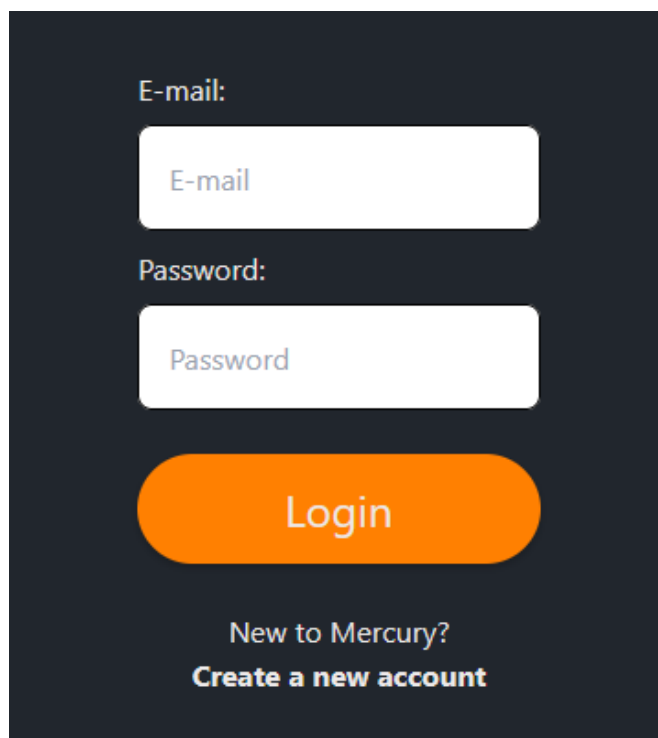
Formularz rejestracyjny

Użytkownik, aby móc się zarejestrować, musi podać swoje dane, które są w odpowiedni sposób walidowane. Reguły w formularzu rejestracji:

1. Imię i nazwisko z przynajmniej 2 znakami
2. Kod państwa o dokładnie 2 literach
3. Brak zajętego konta o tym samym mailu
4. Email o poprawnym formacie
5. Hasło z przynajmniej 8 znakami

Zdjęcie profilowe nie jest wymagane. W przypadku braku wprowadzenia własnego zdjęcia, system sam wprowadza domyślne zdjęcie. Wybór zdjęcia jest widoczny jako miniaturka, również z możliwością usunięcia.

Dane z formularza są walidowane w czasie rzeczywistym przy użyciu biblioteki Zod i React hooka useForm. Po wypełnieniu formularza i wciśnięciu przycisku 'Register' przesłany obrazek (lub w przypadku braku, to obrazek domyślny) jest przekodowywany na format base64. Hasło jest szyfrowane za pomocą biblioteki bcrypt. Następnie sprawdzane jest, czy podany maila z formularza nie jest zajęty w systemie. Jeśli jest zajęty, to wyświetlony jest komunikat z tym związany, formularz nie jest czyszczony w celu poprawienia maila. W przypadku poprawności danych, system zapisuje nowego użytkownika oraz następuje zmiana strony na logowanie na endpointcie: <http://localhost:5173/login>

The image shows a login form on a dark background. It features two white input fields: the first is labeled 'E-mail:' and the second is labeled 'Password:'. Below these fields is a large orange rounded button with the text 'Login' in white. At the bottom, there is a link that says 'New to Mercury? Create a new account' in white text.

Formularz logowania

Formularz logowania wymaga podania maila oraz hasła. Po poprawnym podaniu danych jest tworzona sesja z parą tokenów:

1. access token - token o krótkim czasie ważności (900 sekund - 15 minut) przechowywany w sessionStorage
2. refresh token - token o długim czasie ważności (1209600 sekund - 2 tygodnie) przechowywany na serwerze w ramach ciasteczka secure, httpOnly (przechowywany po stronie serwera)

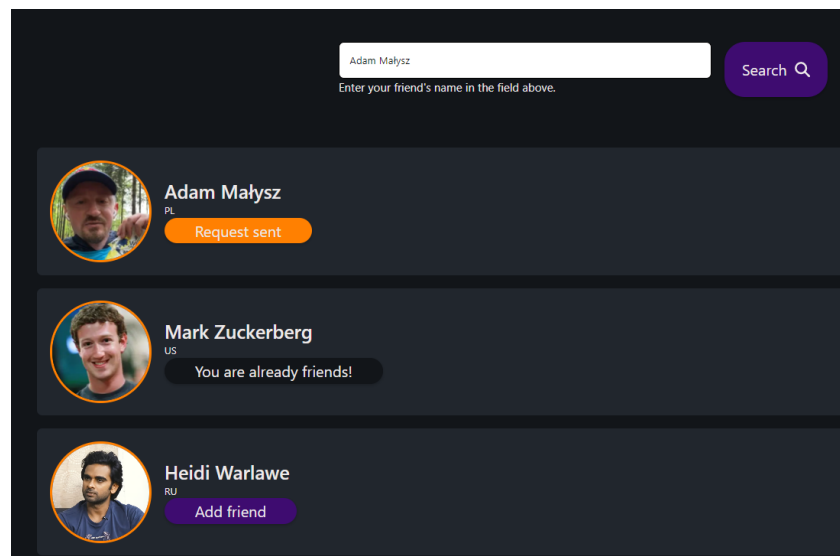
Access token pozwala na dostęp do zasobów. Każdy, kto go posiada ma dobrowolny dostęp do zasobów danego użytkownika. Z tego powodu ważny jest jego

krótki czas ważności. W przypadku utraty ważności klient używa refresh token'a do otrzymania nowego access token'a. Ta operacja jest bezpieczna, ponieważ refresh token jest przechowywany od strony serwera.

Od strony backendu są używane biblioteki cookie-parser i jsonwebtoken a od strony frontendu biblioteka js-cookie. Po zalogowaniu się następuje przekierowanie na URL `http://localhost:5173/profile` do którego ma się tylko dostęp podczas sesji. Zawiera on wszystkie dane, które zostały wcześniej podane w rejestracji.

5.2 System szukania znajomych

Wyszukiwanie znajomych odbywa się na endpointcie `http://localhost:5173/search`. Po wpisaniu w pasek wyszukiwania danej frazy i zatwierdzeniu jest pokazywane 10 pierwszych wyników, które są najbardziej zbliżone do tej frazy na podstawie imienia i nazwiska. W celu wyszukiwania podobieństwa jest wykorzystywany własny algorytm oparty o algorytmie odległości Levenshteina. Każda osoba na liście zawiera przycisk informujący relację z danym kontaktem.



Okno wyszukiwania znajomych

Od strony backendu jest wykorzystany vector search index. Imię i nazwisko są przechowywane jako wektor wartości. Jeżeli jedna para imienia i nazwiska jest podobna do drugiej to odległość między dwoma wektorami powinna być mała. W tym przypadku metryka wykorzystywana do porównania wektorów to cosine similarity.

Do zamiany tekstu na wektor (word embedding) jest wykorzystywany własny algorytm.

Miał on do zrealizowania cele:

1. nie powinien wykorzystywać sztucznej inteligencji (brak danych testowych i długiego trenowania),
2. słowa z literami, które są blisko siebie na układzie QWERTY powinny być podobne,
3. słowa z literami zamienionymi miejscami powinny być podobne,
4. słowa z brakującymi lub dodanymi literami powinny być podobne,
5. część słowa powinna być podobna do całego słowa.

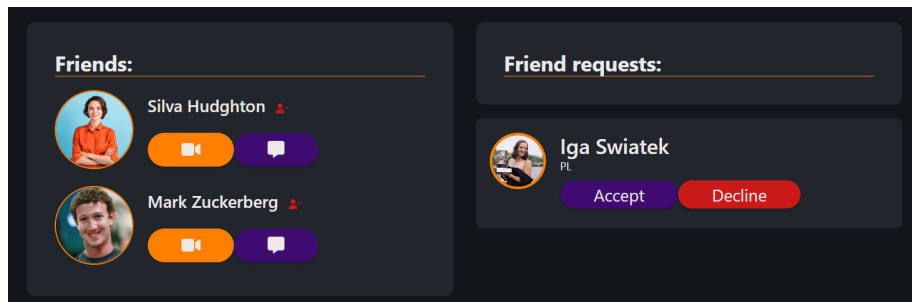
Pierwszym krokiem algorytmu jest zamiana każdej litery na wartość z zakresu $[-1, 1]$. Wartość wskazuje, po której stronie klawiatury znajduje się klawisz, na przykład litera Q ma wartość -1, P ma wartość 1. Porównanie dwóch słów o tej samej długości, ale np. ze zmienioną literą A na Q będzie dawało mały wynik a na przykład z A na L - większy.

Drugim krokiem jest użycie interpolacji liniowej do rozszerzenia wektorów do tej samej, stałej długości. Słowo dłuższe albo krótsze o jedną literę nadal będą się dobrze pokrywać i dadzą bliski wynik.

Trzecim krokiem jest połączenie poprzedniego wyliczonego wektora z wektorem z posortowanymi wartościami. Przy porównywaniu dwóch słów, w tym jednym z zamienionymi literami pierwszy wektor da niższy wynik ale drugi wyższy, co zwiększa odporność algorytmu na zamianę liter.

5.3 System dodawania do znajomych

Po wybraniu kontaktu, który ma zostać dodany do sieci znajomych, wysyłany jest Request i tworzona relacja **SENT_INVITE_TO** w bazie Neo4j. Przychodzące zaproszenia i znajomych można sprawdzić w zakładce *Friends* lub na endpointcie `http://localhost:5173/friends`



Ekran listy znajomych i przychodzących zaproszeń

Zaproszenia do znajomych są wczytywane cały czas na żywo, poprzez wysyłanie requesta do bazy danych o sprawdzeniu relacjach z zalogowanym użytkownikiem. Endpointy wysyłane na bazę danych dla przykładowego id:

1. `/users/userId/friend-requests` - (GET) wczytanie wszystkich zaproszeń do znajomych
2. `/users/userId/friends` - (GET) wczytanie wszystkich obecnych znajomych
3. `/users/userId/remove/friendId` - (DELETE) odrzucenie zaproszenia, usunięcie wszelkich relacji pomiędzy użytkownikiem o id 'userId' oraz 'friendId'
4. `/users/userId/accept/friendId` - (POST) - zaakceptowanie zaproszenia, stworzenie relacji **IS_FRIENDS_WITH** z użytkownikiem o id 'userId' z 'friendId'

5.4 System wideo rozmowy

Użycie WebRTC (*Web Real-Time Communication*) pozwala nawiązać połączenie typu *peer to peer* między dwoma użytkownikami i wymieniać się na bieżąco sygnałem audio i wideo. To rozwiązanie nie potrzebuje serwera, co zmniejsza opóźnienie między użytkownikami. WebRTC transportuje dane z użyciem protokołów UDP, który jest ceniony za swoją szybkość

Schemat komunikacji

1. Gdy Użytkownik1 chce połączyć się z Użytkownikiem2, wysyła on odpowiednią wiadomość o chęci dołączenia.

2. Użytkownik2 chcąc połączyć się, akceptuje 'ofertę' i wysyła Użytkownikowi1 swoje informacje.
3. Po tym, gdy użytkownicy wymieniają się danymi, nawiązuje się połączenie, każdy użytkownik zna SDP drugiego.
4. Używając metody ICE, każdy użytkownik uderza do *stun server* by uzyskać swój publiczny adres IP.
5. Gdy *stun server* odpowie, odebrane dane są poprzez użytkownika transportowane do drugiego. Tak samo działa to u drugiego użytkownika.
6. Kiedy dane znajdują wspólną drogę komunikacji, połączenie jest już gotowe i mogą być przesyłane informacje w obie strony.

Protokół WebRTC zajmuje się tylko przesyłaniem strumieni audio i wideo. Do przesyłania komunikatów o dołączeniu wykorzystuje się inny protokół - w tym projekcie komunikacja odbywa się przez web sockety. Organizacją procesu zajmuje się ICE (*Interactive Connectivity Establishment*) signaling.

W danych, które są zawierane podczas dołączenia do rozmowy między użytkownikami, znajduje się protokół SDP (*Session Description Protocol*), który zawiera informacje typu: kodek, adres, typ nośnika, dane audio, dane wideo. Użytkownicy również wymieniają się '*ICE candidates*', którym jest publiczny adres IP i port, który przyjmuje dostarczane dane.

Od strony backendu spotkania przechowywane są w formie węzłów o etykiecie Meeting. Po dołączeniu do spotkania utworzona jest relacja:

(:User) - [:IS_IN_MEETING] -> (:Meeting).

Przy dowolnej próbie wyjścia ze spotkania (wciśnięcie przycisku do wyjścia ze spotkania, przejście na inną stronę, zamknięcie karty w przeglądarce) jest usuwana relacja i węzeł ze spotkaniem, gdy nie ma żadnych innych relacji.

5.5 System chatu

Chat ze znajomymi jest dostępny poprzez wybranie zielonego przycisku chatu obok znajomego w liście dostępnych znajomych. Nowy chat jest tworzony wraz z przypisanym id. Aby wysłać wiadomość, należy wcisnąć Enter.

Wiadomości wysyłane są w czasie rzeczywistym, a cała konwersacja pomiędzy użytkownikami jest zapisywana do bazy MongoDB. Do komunikacji wykorzystywane są gniazda WebSocket. Historia wiadomości jest dostępna pod endpointem: `/chat/userId/friendId`



Przykładowy ekran chatu

5.6 WebSocket

Połączenia WebSocket przechowywane w bazie danych w formie węzłów o etykiecie Socket. Po zalogowaniu się utworzona jest relacja

`(:User) - [:CONNECTED_TO] -> (:Socket)`.

Przy dowolnej próbie zamknięcia połączenia (wylogowanie się, zamknięcie karty w przeglądarce) jest usuwana relacja i węzeł z socketem, gdy nie ma żadnych innych relacji.

6 Źródła

- Dokumentacja Framer-motion: <https://www.framer.com/motion/>
- Dokumentacja Zod: <https://zod.dev/>
- Dokumentacja Bcrypt: <https://www.npmjs.com/package/bcrypt>
- Dokumentacja Express: <https://expressjs.com/en/api.html>
- Kurs WebRTC: <https://www.youtube.com/watch?v=QsH8FL0952k>