

Sprawozdanie z tematu I – ćwiczenie III

Przedmiot: Systemy urządzeń mobilnych

Autor: Jakub Opiełka

Kierunek: Informatyka dla inżynierów

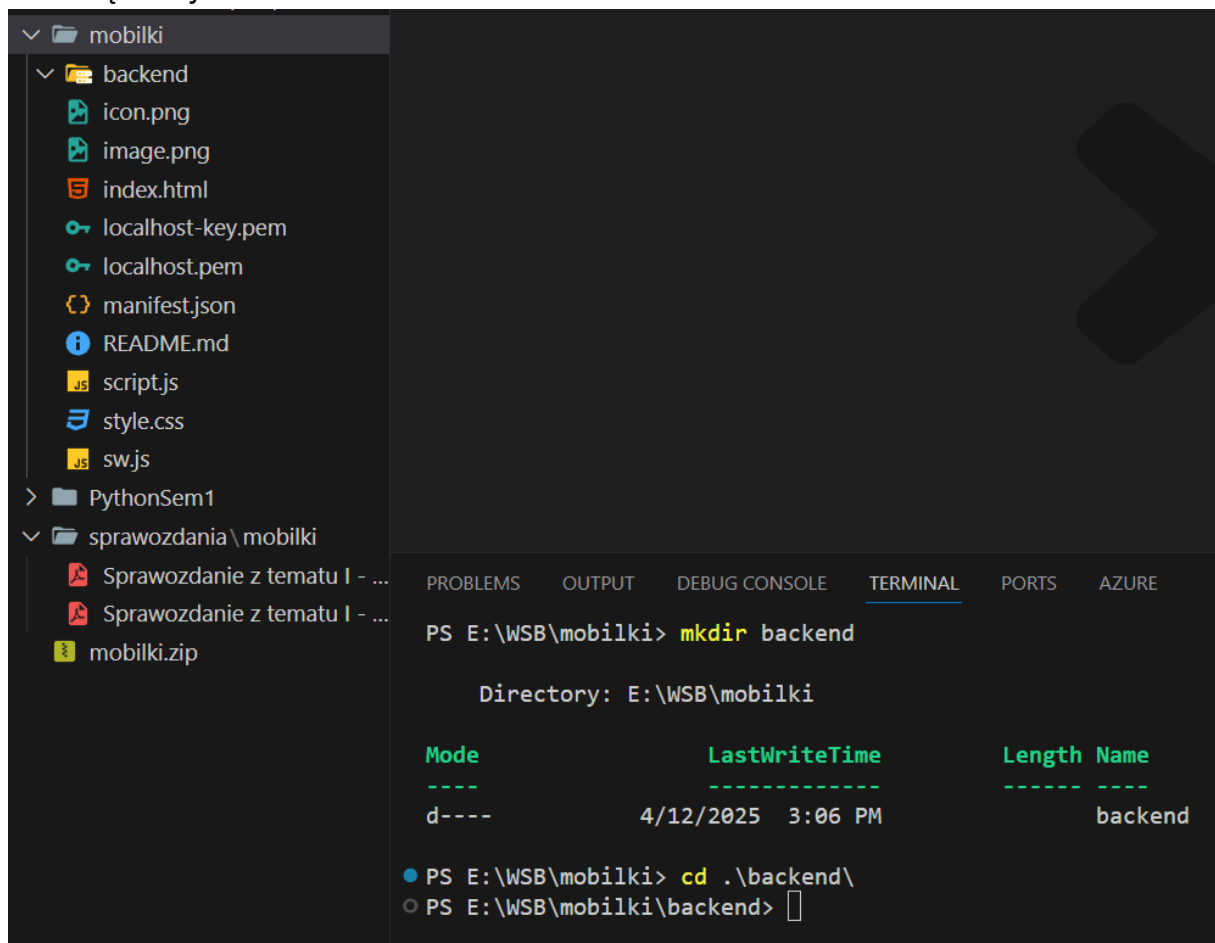
Semestr: I

Krok 1.

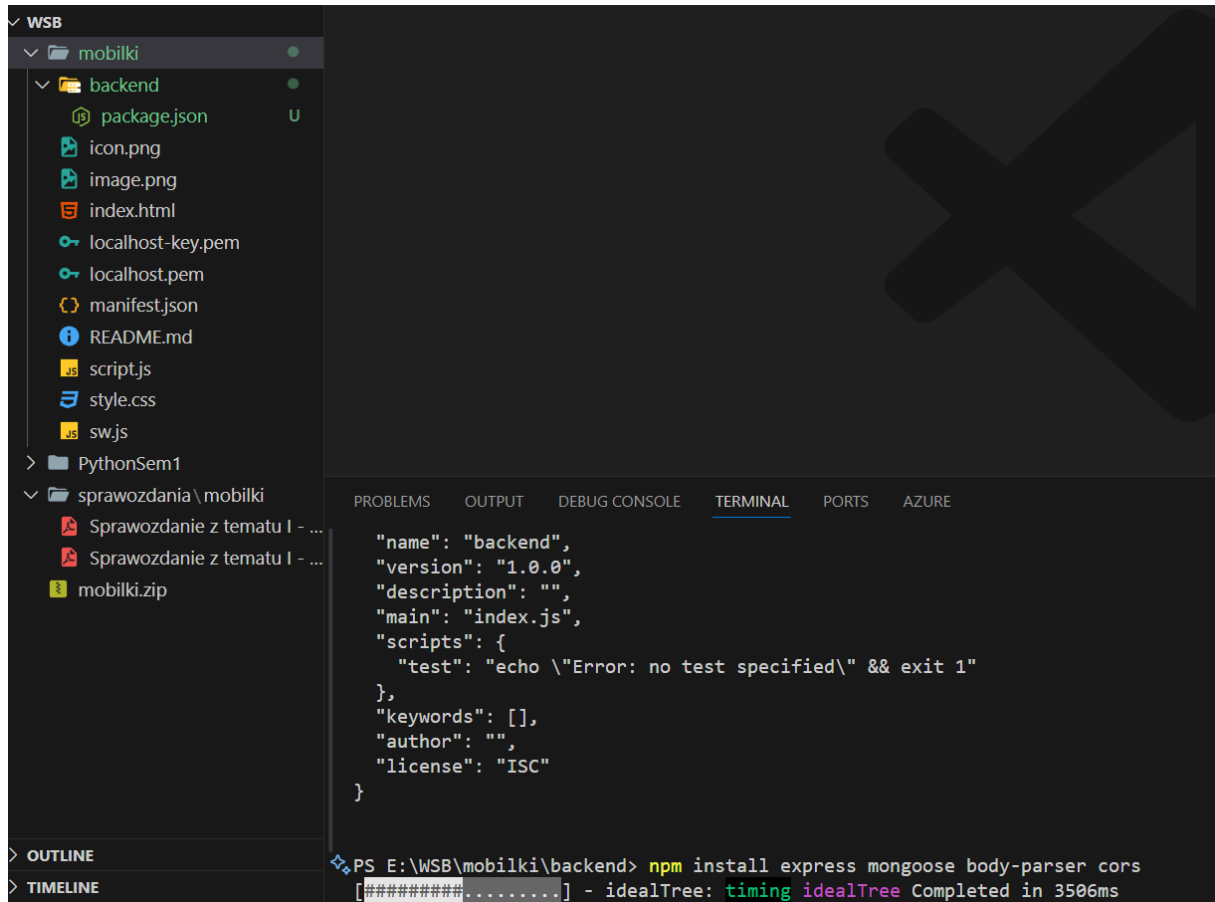
Konfiguracja serwera backendowego.

Tworzę nowy projekt, który będzie moim backendem:

Tworzę nowy folder.



Wykonuje komendy potrzebne do zainicjowania naszego backendu.

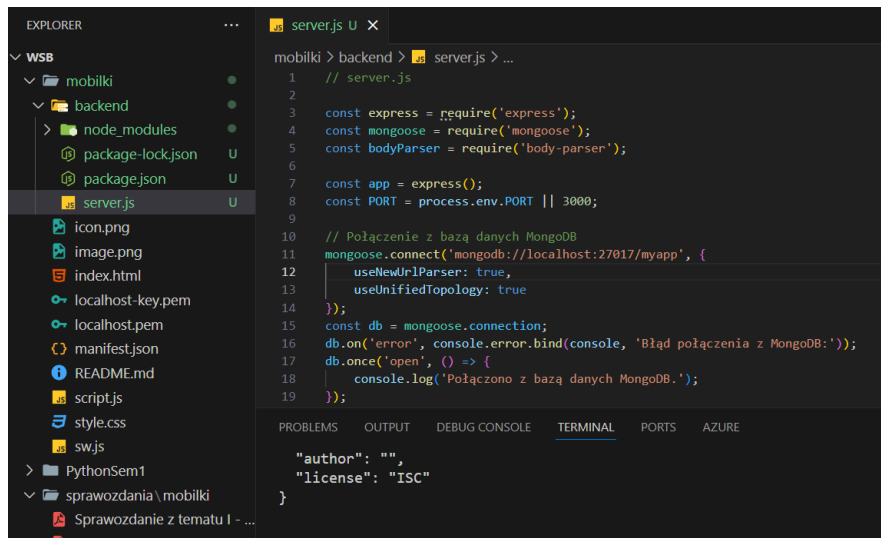


Wyjaśnienie:

- express – serwer HTTP
- mongoose – łączenie z MongoDB
- body-parser – do odczytywania JSONa z POSTów
- cors – potrzebne, bo frontend (localhost:8080) i backend (localhost:3000) to różne porty

Krok 2.

Tworzenie server.js używając zawartości dołączonego instrukcji pliku backendServer.js:

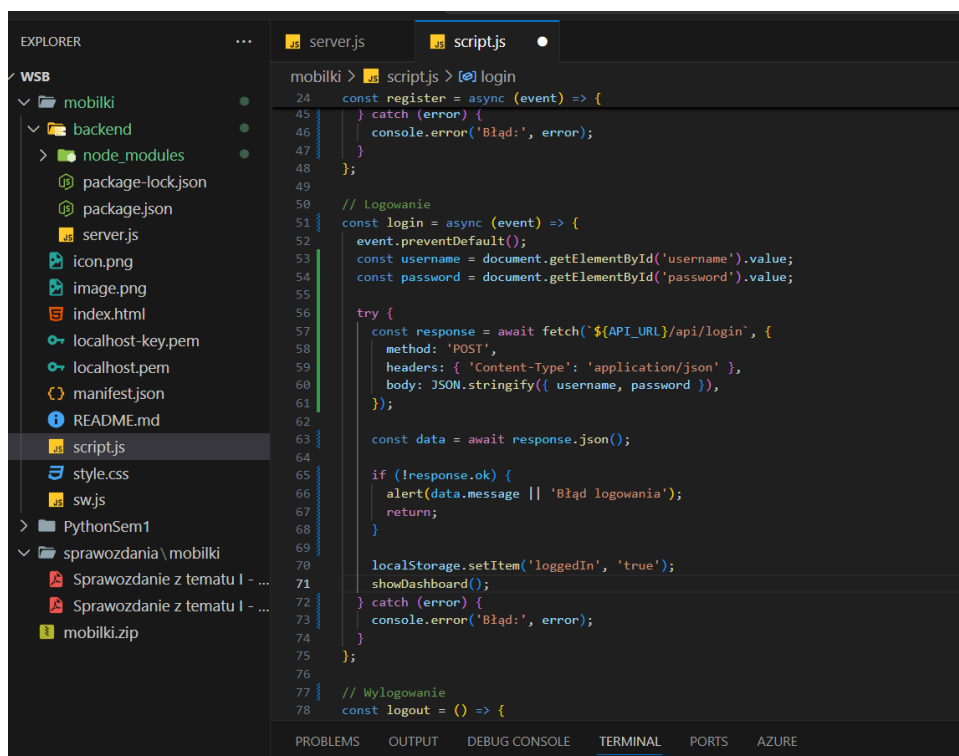


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a file tree for a project named 'mobilki'. The 'backend' folder is expanded, showing 'node_modules', 'package-lock.json', 'package.json', and 'server.js'. The 'server.js' file is selected. The main editor area shows the content of 'server.js', which is a JavaScript file for connecting to a MongoDB database and setting up an Express.js server. The code includes imports for 'express', 'mongoose', and 'body-parser', followed by the setup of the server and database connection. The bottom panel shows the 'TERMINAL' tab with the command 'npm start' and its output.

```
1 // server.js
2
3 const express = require('express');
4 const mongoose = require('mongoose');
5 const bodyParser = require('body-parser');
6
7 const app = express();
8 const PORT = process.env.PORT || 3000;
9
10 // Połączenie z bazą danych MongoDB
11 mongoose.connect('mongodb://localhost:27017/myapp', {
12   useNewUrlParser: true,
13   useUnifiedTopology: true
14 });
15 const db = mongoose.connection;
16 db.on('error', console.error.bind(console, 'Błąd połączenia z MongoDB:'));
17 db.once('open', () => {
18   console.log('Połączono z bazą danych MongoDB.');
```

Krok 3.

Dostosowanie istniejącego skryptu js tak aby mógł obsługiwać rejestrację i logowanie użytkownika.

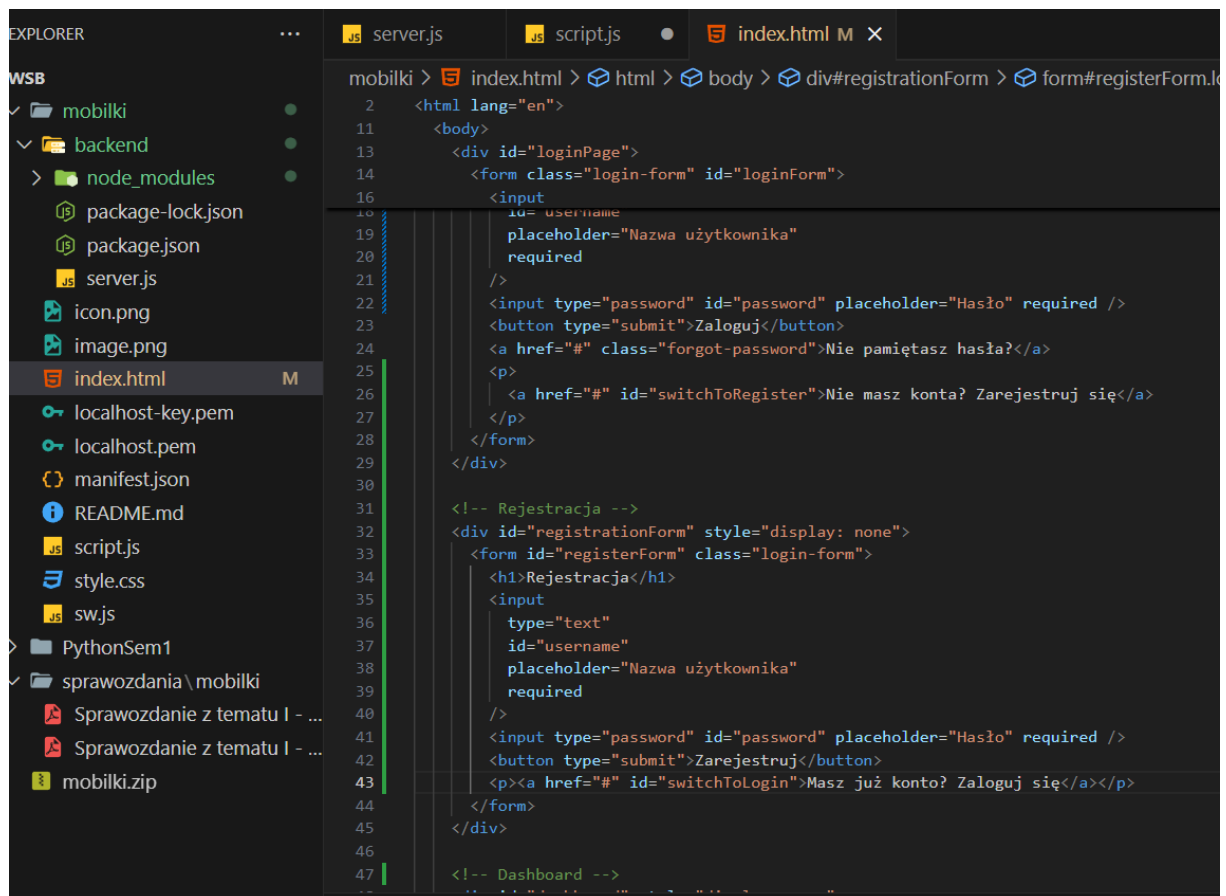


The screenshot shows the Visual Studio Code interface with the 'script.js' file open in the editor. The file is located in the 'mobilki' project, under the 'backend' folder. The code in 'script.js' implements a login function. It uses 'fetch' to send a POST request to the '/api/login' endpoint. The request body contains the username and password. The response is parsed as JSON, and the 'message' property is used to display an alert. The 'login' function is wrapped in a try-catch block. The 'logout' function is also defined at the bottom of the file. The bottom panel shows the 'TERMINAL' tab with the command 'npm start' and its output.

```
24 const register = async (event) => {
25   // Rejestracja
26   event.preventDefault();
27   const username = document.getElementById('username').value;
28   const password = document.getElementById('password').value;
29
30   try {
31     const response = await fetch(`${API_URL}/api/register`, {
32       method: 'POST',
33       headers: { 'Content-Type': 'application/json' },
34       body: JSON.stringify({ username, password })
35     });
36     const data = await response.json();
37     if (!response.ok) {
38       alert(data.message || 'Błąd rejestracji');
39       return;
40     }
41     localStorage.setItem('loggedIn', 'true');
42     showDashboard();
43   } catch (error) {
44     console.error('Błąd:', error);
45   }
46 }
47
48 // Logowanie
49 const login = async (event) => {
50   event.preventDefault();
51   const username = document.getElementById('username').value;
52   const password = document.getElementById('password').value;
53
54   try {
55     const response = await fetch(`${API_URL}/api/login`, {
56       method: 'POST',
57       headers: { 'Content-Type': 'application/json' },
58       body: JSON.stringify({ username, password })
59     });
60     const data = await response.json();
61     if (!response.ok) {
62       alert(data.message || 'Błąd logowania');
63       return;
64     }
65     localStorage.setItem('loggedIn', 'true');
66     showDashboard();
67   } catch (error) {
68     console.error('Błąd:', error);
69   }
70 }
71
72 // Wylogowanie
73 const logout = () => {
74   // Wylogowanie
75   localStorage.removeItem('loggedIn');
76   showDashboard();
77 }
78
```

Krok 4.

Dostosowanie istniejącego pliku index.html tak aby mógł obsługiwać rejestrację i logowanie użytkownika.



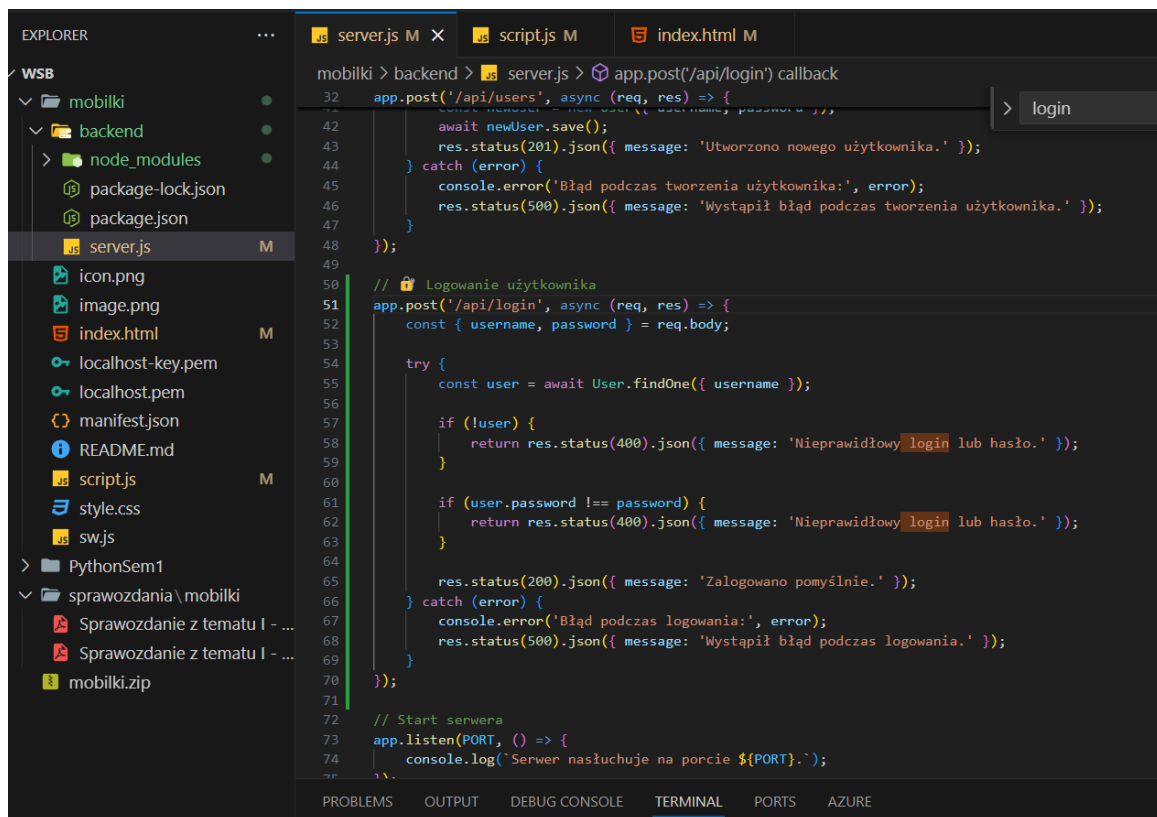
```
2 <html lang="en">
11 <body>
13 <div id="loginPage">
14 <form class="login-form" id="loginForm">
16 <input
18   id="username"
19   placeholder="Nazwa użytkownika"
20   required
21 />
22 <input type="password" id="password" placeholder="Hasło" required />
23 <button type="submit">Zaloguj</button>
24 <a href="#" class="forgot-password">Nie pamiętasz hasła?</a>
25 <p>
26   <a href="#" id="switchToRegister">Nie masz konta? Zarejestruj się</a>
27 </p>
28 </form>
29 </div>
30
31 <!-- Rejestracja -->
32 <div id="registrationForm" style="display: none">
33 <form id="registerForm" class="login-form">
34 <h1>Rejestracja</h1>
35 <input
36   type="text"
37   id="username"
38   placeholder="Nazwa użytkownika"
39   required
40 />
41 <input type="password" id="password" placeholder="Hasło" required />
42 <button type="submit">Zarejestruj</button>
43 <p><a href="#" id="switchToLogin">Masz już konto? Zaloguj się</a></p>
44 </form>
45 </div>
46
47 <!-- Dashboard -->
```

Jakie zmiany zaszły w obu plikach?

Zmiana	Opis
✓ Dodano <code>#registrationForm</code>	Formularz do rejestracji obok logowania
✓ Obsługa <code>/api/users</code>	Wysłał POST do backendu z nazwą i hasłem
✓ Login działa z backendem	Sprawdza dane przez POST <code>/api/login</code> (zakładamy taki endpoint — łatwo go dorobić)
✓ Jeden zestaw pól	<code>username</code> i <code>password</code> są współdzielone
✓ Przejścia widoków	Pokazuje <code>loginPage</code> , <code>registrationForm</code> i <code>dashboard</code> zgodnie ze stanem

Krok 5.

Póki co nasz backend nie jest kompatybilny z frontendem, należy dostosować plik server.js aby obsłużył nasze zapytania.



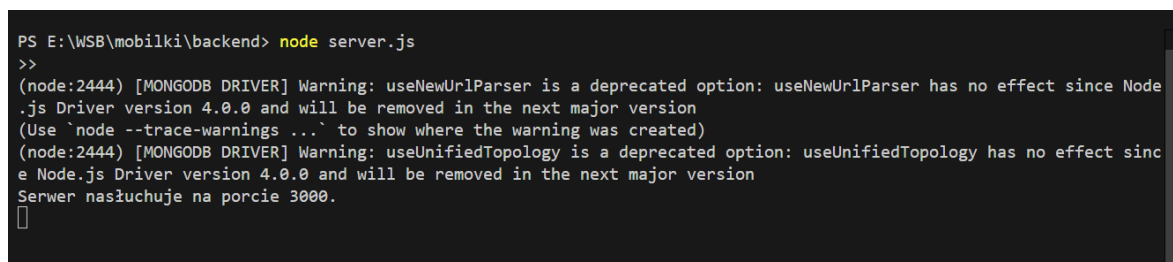
```
server.js
32 app.post('/api/users', async (req, res) => {
33   const { username, password } = req.body;
34   // Tworzenie nowego użytkownika
35   await newUser.save();
36   res.status(201).json({ message: 'Utworzono nowego użytkownika.' });
37 } catch (error) {
38   console.error('Błąd podczas tworzenia użytkownika:', error);
39   res.status(500).json({ message: 'Wystąpił błąd podczas tworzenia użytkownika.' });
40 }
41 });
42
43 // Logowanie użytkownika
44 app.post('/api/login', async (req, res) => {
45   const { username, password } = req.body;
46
47   try {
48     const user = await User.findOne({ username });
49
50     if (!user) {
51       return res.status(400).json({ message: 'Nieprawidłowy login lub hasło.' });
52     }
53
54     if (user.password !== password) {
55       return res.status(400).json({ message: 'Nieprawidłowy login lub hasło.' });
56     }
57
58     res.status(200).json({ message: 'Zalogowano pomyślnie.' });
59   } catch (error) {
60     console.error('Błąd podczas logowania:', error);
61     res.status(500).json({ message: 'Wystąpił błąd podczas logowania.' });
62   }
63 });
64
65 // Start serwera
66 app.listen(PORT, () => {
67   console.log(`Serwer nasłuchuje na porcie ${PORT}.`);
68 });
```

Co zostało zrobione?

- Endpoint POST /api/login
- Sprawdzenie, czy użytkownik istnieje
- Porównanie hasła
- Zwrócenie komunikatu sukcesu lub błędu

Krok 6. – Uruchomienie serwera backendowego oraz frontendowego.

Backend:



```
PS E:\WSB\mobilki\backend> node server.js
>>
(node:2444) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(Use `node --trace-warnings ...` to show where the warning was created)
(node:2444) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
Serwer nasłuchuje na porcie 3000.
```

Frontend:

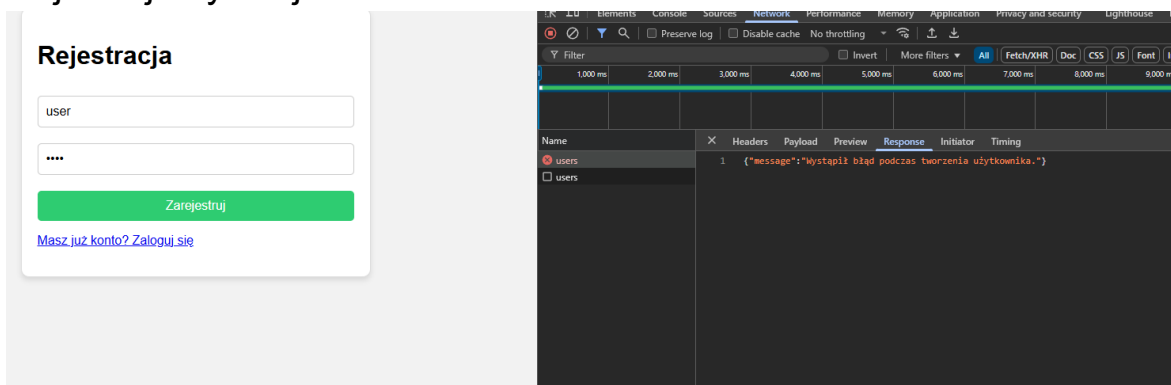
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE

Connection Timeout: 120 seconds
Directory Listings: visible
AutoIndex: visible
Serve GZIP Files: false
Serve Brotli Files: false
Default File Extension: none

Available on:
  https://172.20.240.1:8080
  https://192.168.1.11:8080
  https://127.0.0.1:8080
Hit CTRL-C to stop the server
```

Krok 6. – Test.

Rejestrujemy się jako user:user



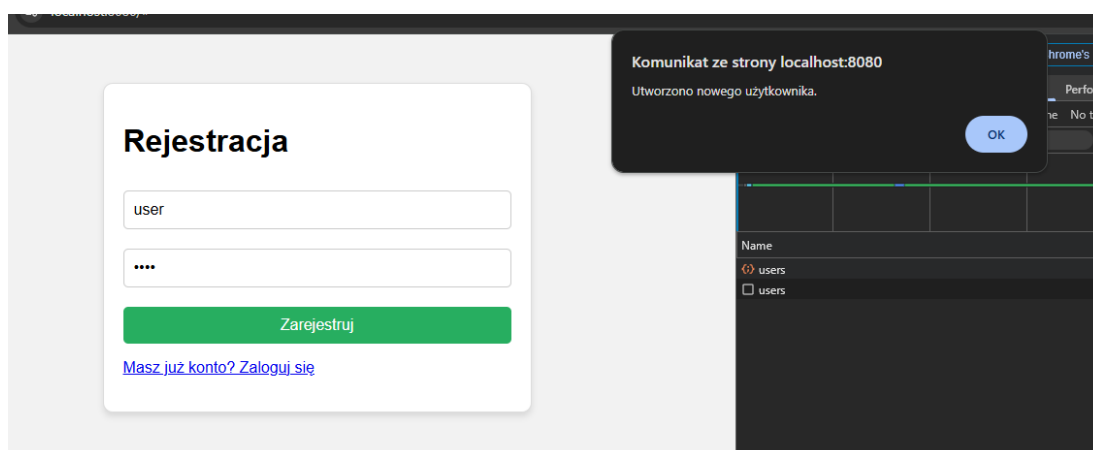
Niestety nadal nie działa, szukamy przyczyny.

Przyczyna: Serwer Node.js nie może połączyć się z lokalną bazą danych MongoDB na porcie 27017

Uruchamiam mongo jako kontener dockerowy na wspomnianym porcie:

```
PS E:\WSB> docker run --name mongoddb -d -p 27017:27017 mongo
>>
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
2726e237d1a3: Downloading [=====] 23.73MB/29.72MB
adf7a2b7606d: Download complete
1a08ed22982f: Download complete
90ade5227b11: Download complete
66eecf3d03b3: Download complete
8fc21ef9255d: Download complete
29fee425ac1e: Downloading [==>] 11.29MB/257.9MB
7e4d144a0672: Download complete
[]
```

Udało się utworzyć nowego użytkownika user:user



Użytkownik został zalogowany do systemu pomyślnie:

