Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 1

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Projektowanie i budowa szkieletu aplikacji internetowej stanowi jeden z kluczowych etapów procesu tworzenia nowoczesnego oprogramowania webowego. Na tym etapie definiowana jest podstawowa struktura projektu, sposób organizacji plików oraz konfiguracja środowiska pracy, co ma bezpośredni wpływ na dalszy rozwój aplikacji, jej czytelność, skalowalność oraz łatwość utrzymania. Odpowiednio zaprojektowany szkielet aplikacji pozwala zespołowi programistycznemu pracować w sposób uporządkowany, minimalizuje ryzyko błędów oraz ułatwia wdrażanie dobrych praktyk programistycznych.

Aplikacje internetowe składają się z wielu współpracujących ze sobą elementów, takich jak warstwa prezentacji (frontend), logika aplikacji oraz konfiguracja środowiska uruchomieniowego. Już na etapie tworzenia szkieletu należy podjąć decyzje dotyczące używanych technologii, struktury katalogów, sposobu zarządzania zależnościami oraz narzędzi wspomagających proces wytwarzania oprogramowania. Coraz częściej wykorzystuje się w tym celu narzędzia automatyzujące konfigurację projektu, takie jak menedżery pakietów, bundlery czy serwery deweloperskie.

Właściwe przygotowanie środowiska pracy jest równie istotne jak sama struktura projektu. Obejmuje ono instalację niezbędnego oprogramowania, konfigurację edytora kodu, ustawienie narzędzi do kontroli wersji oraz przygotowanie mechanizmów uruchamiania i testowania aplikacji. Dzięki temu programista może skupić się na implementacji funkcjonalności, zamiast tracić czas na rozwiązywanie problemów konfiguracyjnych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest nauczenie się tworzenia podstawowego szkieletu aplikacji internetowej, obejmującego logiczną strukturę katalogów oraz niezbędne pliki konfiguracyjne. Student pozna etapy przygotowania środowiska pracy oraz uruchomienia projektu w trybie deweloperskim.

Instrukcja laboratoryjna

Wymagania wstępne

Przed rozpoczęciem ćwiczenia należy upewnić się, że na komputerze zainstalowane są:

- system kontroli wersji Git,
- środowisko uruchomieniowe Node.js (wraz z menedżerem pakietów npm),
- edytor kodu źródłowego (np. Visual Studio Code),
- przeglądarka internetowa (Chrome, Firefox lub inna nowoczesna przeglądarka).

Node.js jest wykorzystywany w ćwiczeniu jako narzędzie do zarządzania zależnościami projektu oraz do uruchamiania serwera deweloperskiego. Git umożliwia śledzenie zmian w projekcie i stanowi standardowe narzędzie pracy zespołowej.

Ćwiczenie 1

Utwórz katalog projektu

1. Na dysku lokalnym utwórz nowy katalog projektu, np. web-app-szkielet.
2. Otwórz edytor kodu i załaduj do niego utworzony katalog projektu.
3. Sprawdź, czy katalog jest pusty i nie zawiera zbędnych plików.

Wynik: pusty katalog projektu otwarty w edytorze kodu, gotowy do inicjalizacji.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 2

Inicjalizacja projektu i konfiguracja podstawowa

1. Otwórz terminal w katalogu projektu.
2. Zainicjalizuj projekt za pomocą menedżera pakietów npm:

```
npm init
```

3. Podczas inicjalizacji uzupełnij podstawowe informacje:
   - nazwę projektu,
   - wersję początkową,
   - krótki opis,
   - punkt wejścia aplikacji.

Po zakończeniu procesu w katalogu projektu pojawi się plik package.json, który zawiera informacje o projekcie, jego zależnościach oraz skryptach uruchomieniowych.

Przykładowa zawartość pliku `package.json`:

```
{
     "name": "web-app-szkielet",
     "version": "1.0.0",
     "description": "Szkielet aplikacji internetowej",
     "scripts": {
          "start": "live-server public",
          "dev": "live-server public"
     }
}
```

Wynik: poprawnie zainicjalizowany projekt z podstawowym plikiem konfiguracyjnym.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 3

Organizacja struktury katalogów projektu

W katalogu głównym projektu utwórz następującą strukturę katalogów:

- `src/` – kod źródłowy aplikacji,
- `public/` – pliki udostępniane bezpośrednio przeglądarce,
- `config/` – pliki konfiguracyjne projektu,
- `dist/` – katalog na pliki wynikowe (generowany automatycznie).

W katalogu src utwórz dodatkowe podkatalogi:

- `js/` – pliki JavaScript odpowiedzialne za logikę aplikacji,
- `css/` – arkusze stylów CSS,
- `assets/` – zasoby statyczne (obrazy, ikony).

Wynik: czytelna i logiczna struktura katalogów ułatwiająca dalszy rozwój aplikacji.

Ćwiczenie 4

Przygotowanie plików startowych aplikacji

1. W katalogu public utwórz plik index.html.
2. W katalogu src/js utwórz plik main.js.
3. W katalogu src/css utwórz plik style.css.

Przykładowa zawartość pliku `index.html`:

```
<!DOCTYPE html>
<html lang="pl">
<head>
     <meta charset="UTF-8">
     <title>Szkielet aplikacji</title>
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    <link rel="stylesheet" href="../src/css/style.css">
</head>
<body>
    <h1>Moja aplikacja internetowa</h1>
    <p>Szkielet projektu działa poprawnie.</p>

    <script src="../src/js/main.js"></script>
</body>
</html>
```

Przykładowa zawartość pliku `main.js`:

```
document.addEventListener('DOMContentLoaded', () => {
    console.log('Aplikacja została uruchomiona');
});
```

Przykładowa zawartość pliku `style.css`:

```
body {
    font-family: Arial, sans-serif;
    margin: 40px;
    background-color: #f5f5f5;
}
```

Wynik: aplikacja posiada komplet minimalnych plików umożliwiających jej uruchomienie w przeglądarce.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 5

Konfiguracja środowiska deweloperskiego

1. Zainstaluj lokalnie prosty serwer deweloperski:

```
npm install --save-dev live-server
```

2. Sprawdź, czy w pliku `package.json` znajdują się skrypty umożliwiające uruchomienie projektu.

3. Uruchom aplikację w trybie deweloperskim:

```
npm run dev
```

Serwer automatycznie otworzy aplikację w przeglądarce i będzie reagował na zmiany w plikach.

Wynik: działające środowisko deweloperskie z automatycznym odświeżaniem strony.

Ćwiczenie 6

Integracja z systemem kontroli wersji

1. Zainicjalizuj repozytorium Git:

```
git init
```

2. Utwórz plik `.gitignore` i dodaj do niego katalogi, które nie powinny być śledzone:

```
node_modules/
dist/
```

3. Dodaj pliki do repozytorium i wykonaj pierwsze zatwierdzenie:

```
git add .
git commit -m "Inicjalny szkielet aplikacji"
```

Wynik: projekt znajduje się pod kontrolą wersji i jest gotowy do dalszego rozwoju.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 7

Sprawdzenie poprawności działania projektu

1. Uruchom serwer deweloperski.
2. Otwórz aplikację w przeglądarce.
3. Sprawdź poprawność wyświetlania treści strony.
4. Zweryfikuj, czy w konsoli przeglądarki pojawia się komunikat z pliku JavaScript.

Wynik końcowy: poprawnie działający szkielet aplikacji internetowej, przygotowany do dalszej rozbudowy.

Ćwiczenia samodzielne

Ćwiczenie 1
Zmodyfikuj strukturę katalogów, dodając osobny katalog na komponenty interfejsu użytkownika.

Ćwiczenie 2
Skonfiguruj alternatywne narzędzie do uruchamiania serwera deweloperskiego.

Ćwiczenie 3
Dodaj plik konfiguracyjny środowiska produkcyjnego.

Ćwiczenie 4
Rozszerz plik ignorujący o dodatkowe reguły.

Ćwiczenie 5
Przygotuj prostą stronę informacyjną w ramach istniejącego szkieletu.

Ćwiczenie 6
Dodaj drugi arkusz stylów i podłącz go do projektu.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Pytania pomocnicze

1. Jaką rolę pełni szkielet aplikacji internetowej w procesie jej rozwoju?
2. Dlaczego logiczna struktura katalogów jest istotna w większych projektach?
3. Jakie elementy obejmuje konfiguracja środowiska pracy?
4. W jaki sposób system kontroli wersji wspiera pracę zespołową?
5. Jakie są zalety korzystania z serwera deweloperskiego?
6. Które elementy projektu powinny być ignorowane przez system kontroli wersji?
7. Jak przygotowany szkielet wpływa na późniejszą skalowalność aplikacji?

Podsumowanie

W ramach ćwiczenia zrealizowano proces projektowania i budowy podstawowego szkieletu aplikacji internetowej, obejmujący utworzenie logicznej struktury projektu, przygotowanie plików startowych oraz konfigurację środowiska deweloperskiego. Osiągniętym celem dydaktycznym było nabycie umiejętności organizacji projektu webowego, inicjalizacji środowiska pracy oraz świadomego korzystania z narzędzi wspierających rozwój aplikacji, takich jak menedżer pakietów i system kontroli wersji. Realizacja ćwiczenia pozwala studentowi zrozumieć znaczenie dobrego przygotowania projektu przed rozpoczęciem implementacji właściwej funkcjonalności oraz wpływ tych decyzji na czytelność i rozwijalność kodu.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Warto zastanowić się, które etapy ćwiczenia sprawiły największą trudność, jakie problemy konfiguracyjne mogły się pojawić oraz w jaki sposób można by usprawnić zaproponowaną strukturę projektu. Zdobyta wiedza ma bezpośrednie zastosowanie praktyczne w pracy programisty frontendowego lub full-stack, gdzie poprawne zaprojektowanie szkieletu aplikacji stanowi standardowy i niezbędny element procesu wytwarzania oprogramowania.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 2

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Tematem niniejszej instrukcji jest struktura i tworzenie podstawowego dokumentu HTML, wprowadzenie do protokołu HTTP oraz przechwytywanie i analiza żądań sieciowych. Zagadnienia te stanowią fundament współczesnych aplikacji internetowych i są niezbędne do zrozumienia sposobu działania stron WWW oraz komunikacji pomiędzy przeglądarką a serwerem.

HTML (HyperText Markup Language) jest językiem znaczników służącym do opisu struktury dokumentu internetowego. Za jego pomocą definiuje się elementy takie jak nagłówki, akapity, listy, formularze czy sekcje logiczne strony. Poprawnie zbudowany dokument HTML jest punktem wyjścia do dalszego rozwoju aplikacji webowej, w tym stylowania (CSS) oraz nadawania interaktywności (JavaScript).

Integralnym elementem funkcjonowania stron internetowych jest protokół HTTP (HyperText Transfer Protocol), który odpowiada za komunikację pomiędzy klientem (najczęściej przeglądarką internetową) a serwerem. Protokół ten określa sposób przesyłania żądań i odpowiedzi, ich strukturę, metody (np. GET, POST) oraz kody statusów informujące o wyniku realizacji żądania.

Zrozumienie mechanizmu działania HTTP umożliwia świadome projektowanie aplikacji internetowych, diagnozowanie problemów oraz optymalizację komunikacji sieciowej. W tym kontekście szczególnie istotna jest umiejętność przechwytywania i analizy żądań HTTP za pomocą narzędzi deweloperskich przeglądarek.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest nauczenie studenta tworzenia poprawnego dokumentu HTML, zrozumienia zasad działania protokołu HTTP oraz zdobycia umiejętności przechwytywania i analizy żądań sieciowych. Student pozna podstawowe elementy struktury strony WWW oraz nauczy się interpretować informacje przesyłane pomiędzy klientem a serwerem.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie środowiska pracy

Do realizacji ćwiczenia wymagany jest:

- dowolny edytor kodu (np. Visual Studio Code),
- przeglądarka internetowa (Chrome, Firefox lub Edge),
- podstawowa znajomość obsługi systemu plików.

Utwórz nowy katalog projektu, np. `html_http_lab`, a następnie w jego wnętrzu utwórz plik `index.html`.

Wynik: przygotowane środowisko pracy oraz struktura projektu umożliwiająca dalszą realizację ćwiczenia.

Ćwiczenie 2

Tworzenie podstawowego dokumentu HTML

Otwórz plik `index.html` i wprowadź podstawową strukturę dokumentu HTML5:

```
<!DOCTYPE html>
<html lang="pl">
<head>
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    <meta charset="UTF-8">
    <title>Moja pierwsza strona HTML</title>
</head>
<body>
    <h1>Witaj na stronie</h1>
    <p>To jest przykładowy dokument HTML.</p>
</body>
</html>
```

Zapisz plik i otwórz go w przeglądarce. Przeglądarka zinterpretuje kod HTML i wyświetli nagłówek oraz akapit tekstu.

Wynik: poprawnie wyświetlona strona HTML zawierająca podstawowe elementy strukturalne.

Ćwiczenie 3

Omówienie struktury dokumentu

- `<!DOCTYPE html>` – deklaracja typu dokumentu,
- `<html>` – element nadrzędny dokumentu,
- `<head>` – sekcja metadanych (kodowanie, tytuł strony),
- `<body>` – właściwa treść strony widoczna dla użytkownika.

Wynik: student rozumie rolę podstawowych znaczników HTML i potrafi wskazać ich funkcję w dokumencie.

Ćwiczenie 4

Dodawanie elementów strukturalnych

Rozszerz dokument o kolejne elementy:

```
<h2>Sekcja informacyjna</h2>
<ul>
```


**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    <li>HTML – struktura</li>
    <li>CSS – wygląd</li>
    <li>JavaScript – interakcja</li>
</ul>
```

Odśwież stronę i zaobserwuj zmiany.

Lista nieuporządkowana (`<ul>`)

1. W pliku `index.html` (wewnątrz `<body>`) dodaj nagłówek i listę punktowaną:

```
<h2>Technologie frontendu</h2>
<ul>
   <li>HTML</li>
   <li>CSS</li>
   <li>JavaScript</li>
</ul>
```

2. Zapisz plik.
3. Odśwież stronę w przeglądarce.

Lista uporządkowana (`<ol>`)

1. Bezpośrednio pod listą nieuporządkowaną dodaj listę numerowaną:

```
<h2>Kroki tworzenia strony</h2>
<ol>
   <li>Utwórz plik HTML</li>
   <li>Zdefiniuj strukturę dokumentu</li>
   <li>Dodaj treść w sekcji body</li>
   <li>Sprawdź wynik w przeglądarce</li>
</ol>
```

2. Zapisz plik.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

3. Odśwież stronę w przeglądarce.

Zagnieżdżanie list

Aby zobaczyć, jak można łączyć listy, zagnieźdź listę `<ul>` wewnątrz jednego elementu `<ol>`:

```
<h2>Plan nauki</h2>
<ol>
   <li>Poznaj podstawy HTML</li>
   <li>Poznaj CSS
      <ul>
           <li>selektory</li>
           <li>kolory i typografia</li>
           <li>układ strony (flex/grid)</li>
      </ul>
   </li>
   <li>Poznaj JavaScript</li>
</ol>
```

Wynik: dokument HTML został rozszerzony o dodatkowe elementy strukturalne, a student potrafi ocenić ich wpływ na układ strony.

Ćwiczenie 5

Obserwacja podstawowego żądania HTTP

1. Otwórz plik `index.html` w przeglądarce internetowej.
2. Uruchom narzędzia deweloperskie (klawisz F12).
3. Przejdź do zakładki Network (Sieć).
4. Odśwież stronę (klawisz F5).

Zadania:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

1. Zidentyfikuj żądanie dotyczące pliku index.html.
2. Sprawdź, jakiej metody HTTP użyto.
3. Odczytaj kod statusu odpowiedzi.

Wynik: student potrafi wskazać podstawowe żądanie HTTP typu GET generowane podczas ładowania strony.

Ćwiczenie 6

Analiza szczegółów żądania i odpowiedzi

Kliknij żądanie index.html na liście żądań. Przeanalizuj zakładki Headers, Response oraz Timing.

1. Odczytaj najważniejsze nagłówki żądania (np. User-Agent, Accept).
2. Odczytaj wybrane nagłówki odpowiedzi (np. Content-Type).
3. Sprawdź czas trwania żądania.

Wynik: student rozumie strukturę żądania i odpowiedzi HTTP oraz potrafi interpretować podstawowe nagłówki.

Ćwiczenie 7

Analiza wielu żądań HTTP

1. Dodaj do dokumentu HTML zewnętrzny zasób (obraz lub arkusz CSS), a następnie przeanalizuj, jakie dodatkowe żądania pojawią się w zakładce Network.

Wariant A – dodanie obrazu (IMG):

1. Utwórz w katalogu projektu folder assets/ (jeśli go nie ma).
2. Skopiuj do niego dowolny plik graficzny, np. logo.png.
3. W pliku index.html dodaj wewnątrz <body> element obrazu:

```
<img src="assets/logo.png" alt="Logo" width="200">
```

4. Zapisz plik i odśwież stronę.

Wariant B – dodanie zewnętrznego arkusza CSS:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

1. Utwórz w katalogu projektu folder styles/ (jeśli go nie ma).
2. Utwórz plik styles/main.css i wpisz przykładową regułę:

```
body {
   font-family: Arial, sans-serif;
}
```

3. W sekcji <head> pliku index.html dodaj odwołanie do arkusza stylów:

```
<link rel="stylesheet" href="styles/main.css">
```

4. Zapisz plik i odśwież stronę.

Zadania:

1. Zlicz liczbę żądań HTTP wygenerowanych podczas ładowania strony.
2. Określ, które żądania dotyczą dokumentu HTML, a które zasobów dodatkowych (obraz/CSS).
3. Porównaj czasy ładowania poszczególnych zasobów w zakładce Timing.

Wynik: student zauważa zależność pomiędzy liczbą zasobów strony a liczbą żądań HTTP oraz potrafi wskazać żądania powiązane z konkretnymi plikami.

Ćwiczenie 8

Interpretacja kodów statusów HTTP

1. Spróbuj odwołać się w przeglądarce do nieistniejącego pliku HTML.
2. Zaobserwuj odpowiedź w zakładce Network.

Zadania:

1. Zidentyfikuj kod statusu odpowiedzi.
2. Wyjaśnij jego znaczenie.

Wynik: student potrafi rozróżniać podstawowe kody statusów HTTP i rozumie ich znaczenie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 9

Symulacja dodatkowych żądań

Dodaj do dokumentu obraz lub arkusz CSS, a następnie ponownie przeanalizuj ruch sieciowy. Zauważ, że każde dodatkowe zasoby generują osobne żądania HTTP.

Ćwiczenia samodzielne

Ćwiczenie 1
Utwórz dokument HTML zawierający nagłówki od <h1> do <h3> oraz kilka akapitów.

Ćwiczenie 2
Dodaj listę uporządkowaną i nieuporządkowaną opisującą elementy strony WWW.

Ćwiczenie 3
Przeanalizuj różnice pomiędzy kodami statusów 200, 404 i 500.

Ćwiczenie 4
Dodaj zewnętrzny arkusz CSS i sprawdź, jakie żądanie HTTP jest generowane.

Ćwiczenie 5
Zidentyfikuj wszystkie nagłówki przesyłane w żądaniu HTTP.

Ćwiczenie 6
Sprawdź, jak zmienia się liczba żądań po dodaniu obrazu.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Pytania pomocnicze

1. Jaką rolę pełni znacznik <!DOCTYPE html>?
2. Czym różni się sekcja <head> od <body>?
3. Na czym polega komunikacja klient–serwer w protokole HTTP?
4. Jakie informacje zawierają nagłówki HTTP?
5. Dlaczego analiza żądań HTTP jest istotna w pracy programisty?

Podsumowanie

W ramach ćwiczenia student zapoznał się z budową podstawowego dokumentu HTML oraz zasadami komunikacji HTTP. Zrealizowane zadania pozwoliły zrozumieć, w jaki sposób przeglądarka interpretuje kod HTML i komunikuje się z serwerem. Zdobyte umiejętności umożliwiają analizę ruchu sieciowego oraz diagnozowanie problemów w aplikacjach webowych. Wiedza ta stanowi podstawę do dalszej nauki technologii frontendowych i backendowych.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 3

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Tematem niniejszej instrukcji jest tworzenie interfejsu użytkownika aplikacji internetowej, projektowanie struktury aplikacji oraz implementacja podstawowych elementów interfejsu użytkownika. Zagadnienia te stanowią jeden z kluczowych obszarów frontendowego etapu tworzenia aplikacji webowych i mają bezpośredni wpływ na użyteczność, czytelność oraz odbiór aplikacji przez użytkownika końcowego.

Interfejs użytkownika (UI – User Interface) to warstwa aplikacji odpowiedzialna za prezentację informacji oraz umożliwienie interakcji pomiędzy użytkownikiem a systemem. Obejmuje on wszystkie widoczne elementy strony, takie jak nagłówki, menu nawigacyjne, formularze, przyciski, listy czy sekcje treści. Poprawnie zaprojektowany interfejs powinien być intuicyjny, spójny wizualnie oraz logicznie uporządkowany, co przekłada się na komfort pracy użytkownika i efektywność korzystania z aplikacji.

Projektowanie struktury aplikacji internetowej polega na logicznym podziale interfejsu na sekcje funkcjonalne oraz na odpowiednim doborze znaczników HTML. Wykorzystanie semantycznych elementów HTML, takich jak `<header>`, `<main>`, `<section>`, `<nav>` czy `<footer>`, zwiększa czytelność kodu, ułatwia jego utrzymanie oraz poprawia dostępność aplikacji, m.in. dla czytników ekranowych. Dobrze zaprojektowana struktura stanowi solidną podstawę do dalszej rozbudowy aplikacji oraz implementacji warstwy stylów i logiki.

Na etapie tworzenia interfejsu użytkownika istotne jest także zachowanie zasady rozdzielenia odpowiedzialności pomiędzy poszczególne warstwy aplikacji. HTML odpowiada za strukturę i semantykę interfejsu, CSS za jego wygląd i układ, natomiast JavaScript – za interakcję i zachowanie dynamiczne. Takie podejście sprzyja tworzeniu kodu modularnego, łatwego w utrzymaniu i skalowalnego.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

W ramach niniejszego laboratorium student zapozna się z podstawowymi zasadami projektowania interfejsu użytkownika oraz z praktyczną implementacją najczęściej wykorzystywanych elementów UI w aplikacjach internetowych. Zdobyta wiedza stanowi fundament do dalszej nauki stylowania interfejsów, pracy z responsywnym designem oraz wykorzystania nowoczesnych frameworków frontendowych.

Cel laboratorium

Celem laboratorium jest zapoznanie studenta z zasadami projektowania struktury aplikacji internetowej oraz implementacji podstawowych elementów interfejsu użytkownika. Student nauczy się tworzyć przejrzysty i logiczny układ strony oraz poprawnie wykorzystywać znaczniki HTML do budowy interfejsu.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie struktury projektu

W pierwszym etapie ćwiczenia należy przygotować podstawową strukturę projektu aplikacji internetowej.

1. Na dysku lokalnym utwórz nowy katalog projektu i nadaj mu nazwę ui_app.
2. Otwórz utworzony katalog w wybranym edytorze kodu.
3. W katalogu głównym projektu utwórz plik index.html, który będzie stanowił główny punkt wejścia aplikacji.
4. (Opcjonalnie) Utwórz podkatalog styles, przeznaczony na arkusze stylów CSS, które będą wykorzystywane na kolejnych etapach pracy.
5. Sprawdź, czy struktura katalogów i plików jest zgodna z założeniami oraz czy plik index.html jest poprawnie zapisany.

Wynik: student posiada poprawnie przygotowaną strukturę projektu aplikacji internetowej, zawierającą plik startowy oraz (opcjonalnie) katalog na pliki stylów, co umożliwia dalszą implementację interfejsu użytkownika.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 2

Tworzenie podstawowego szkieletu dokumentu HTML

W pliku `index.html` wprowadź podstawową strukturę dokumentu HTML:

```
<!DOCTYPE html>
<html lang="pl">
<head>
     <meta charset="UTF-8">
     <title>Interfejs użytkownika aplikacji</title>
</head>
<body>
</body>
</html>
```

Zapisz plik i otwórz go w przeglądarce.

Wynik: poprawnie wyświetlona pusta strona HTML.

Ćwiczenie 3

Projektowanie struktury interfejsu aplikacji

Dodaj do sekcji <body> główne elementy strukturalne aplikacji:

```
<header>
     <h1>Moja aplikacja</h1>
</header>

<main>
     <section>
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
            <h2>Panel główny</h2>
            <p>Treść aplikacji.</p>
        </section>
</main>


<footer>
        <p>&copy; 2026</p>
</footer>
```

Wynik: interfejs aplikacji posiada logiczny podział na nagłówek, część główną i stopkę.

Ćwiczenie 4

Implementacja przycisków i sekcji interakcji

W sekcji `<main>` dodaj przycisk akcji:

```
<button type="button">Kliknij mnie</button>
```

Odśwież stronę i sprawdź wygląd elementu.

Wynik: student potrafi dodać podstawowy element interaktywny do interfejsu użytkownika.

Ćwiczenie 5

Tworzenie formularza użytkownika

Dodaj do dokumentu prosty formularz:

```
<form>
        <label for="name">Imię:</label>
        <input type="text" id="name" name="name">
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    <label for="email">Email:</label>
    <input type="email" id="email" name="email">

    <button type="submit">Wyślij</button>
</form>
```

Odśwież stronę i sprawdź działanie formularza.

Wynik: student potrafi tworzyć formularze oraz rozumie rolę pól wejściowych i przycisków.

Ćwiczenie 6

wykorzystanie list w interfejsie użytkownika

Dodaj listę elementów menu:

```
<nav>
    <ul>
        <li>Strona główna</li>
        <li>O aplikacji</li>
        <li>Kontakt</li>
    </ul>
</nav>
```

Wynik: student potrafi wykorzystać listy HTML do budowy elementów nawigacyjnych.

Ćwiczenie 7

grupowanie elementów interfejsu

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Dodaj kontener grupujący elementy:

```
<div class="panel">
     <h3>Panel użytkownika</h3>
     <p>Status: aktywny</p>
</div>
```

Wynik: student rozumie zastosowanie kontenerów do logicznego grupowania elementów UI.

Ćwiczenie 8

Analiza struktury interfejsu użytkownika w DOM (Document Object Model)

1. Otwórz stronę aplikacji w przeglądarce internetowej.
2. Uruchom narzędzia deweloperskie (klawisz **F12** lub opcja *Zbadaj element*).
3. Przejdź do zakładki **Elements** (lub **Inspektor**).
4. Zlokalizuj w drzewie DOM elementy dodane w poprzednich zadaniach, takie jak:
   - `<header>` z nagłówkiem aplikacji,
   - `<main>` oraz `<section>` z treścią,
   - formularz `<form>` wraz z polami `<input>` i przyciskiem,
   - listy `<ul>` / `<li>` oraz kontenery `<div>`.
5. Kliknij wybrane elementy w drzewie DOM i zaobserwuj, które fragmenty interfejsu są podświetlane na stronie.

Zadania:

1. Porównaj strukturę drzewa DOM z kodem HTML zapisanym w pliku index.html.
2. Zidentyfikuj relacje nadrzędne i podrzędne pomiędzy elementami interfejsu.
3. Wskaż, które elementy pełnią rolę kontenerów strukturalnych, a które elementów treściowych lub interaktywnych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Wynik: student rozumie pojęcie DOM, potrafi analizować strukturę interfejsu użytkownika w narzędziach deweloperskich oraz świadomie powiązać kod HTML z jego reprezentacją w przeglądarce.

Ćwiczenia samodzielne

Ćwiczenie 1
Dodaj drugi formularz (np. formularz logowania).

Ćwiczenie 2
Rozszerz interfejs o dodatkową sekcję informacyjną.

Ćwiczenie 3
Zastosuj listę uporządkowaną do prezentacji kroków działania aplikacji.

Ćwiczenie 4
Zaprojektuj prosty panel boczny aplikacji.

Ćwiczenie 5
Dodaj co najmniej dwa przyciski o różnych funkcjach.

Pytania pomocnicze

1. Jaką rolę pełni interfejs użytkownika w aplikacji internetowej?
2. Dlaczego struktura aplikacji powinna być logiczna i modularna?
3. Jakie znaczenie mają semantyczne znaczniki HTML?
4. Jakie elementy interfejsu zwiększają użyteczność aplikacji?
5. W jaki sposób DOM odzwierciedla strukturę interfejsu?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Podsumowanie

W ramach ćwiczenia zaprojektowano strukturę aplikacji internetowej oraz zaimplementowano podstawowe elementy interfejsu użytkownika. Osiągniętym celem było nabycie umiejętności tworzenia logicznego i czytelnego UI z wykorzystaniem znaczników HTML. Ćwiczenie pozwoliło zrozumieć znaczenie poprawnej struktury aplikacji oraz jej wpływu na dalszy rozwój i utrzymanie projektu. Zdobyta wiedza stanowi podstawę do dalszej pracy z dynamicznymi interfejsami oraz frameworkami frontendowymi.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 4

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Tematem niniejszej instrukcji jest dodawanie stylów do interfejsu użytkownika aplikacji internetowej, wykorzystanie kaskadowych arkuszy stylów (CSS) w projektowaniu wizualnym oraz dostosowanie aplikacji do różnych urządzeń i rozdzielczości ekranów. Stylowanie interfejsu użytkownika stanowi kluczowy etap procesu tworzenia aplikacji webowych, ponieważ bezpośrednio wpływa na czytelność, estetykę oraz użyteczność strony.

CSS (Cascading Style Sheets) jest językiem służącym do opisu wyglądu dokumentów HTML. Pozwala on na definiowanie takich cech elementów interfejsu jak kolory, czcionki, odstępy, obramowania, układ elementów czy reakcja na zmiany rozmiaru ekranu. Dzięki rozdzieleniu warstwy strukturalnej (HTML) od warstwy prezentacyjnej (CSS) możliwe jest tworzenie bardziej przejrzystego, łatwego w utrzymaniu i skalowalnego kodu.

Projektowanie wizualne interfejsu użytkownika obejmuje nie tylko dobór kolorów i typografii, ale również odpowiednie rozmieszczenie elementów na stronie. W tym celu stosuje się mechanizmy układu takie jak model pudełkowy (box model), Flexbox czy Grid Layout. Ich poprawne wykorzystanie umożliwia budowanie przejrzystych i elastycznych interfejsów użytkownika.

Istotnym aspektem nowoczesnych aplikacji internetowych jest ich responsywność, czyli zdolność do poprawnego wyświetlania się na różnych urządzeniach – komputerach stacjonarnych, tabletach oraz smartfonach. W tym celu stosuje się m.in. jednostki względne, media queries oraz elastyczne układy. Zrozumienie tych mechanizmów pozwala tworzyć aplikacje dostosowane do współczesnych standardów i oczekiwań użytkowników.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Cel laboratorium

Celem laboratorium jest zapoznanie studenta z podstawami stylowania interfejsu użytkownika przy użyciu CSS oraz z zasadami projektowania wizualnego aplikacji internetowych. Student nauczy się podłączać arkusze stylów, definiować podstawowe reguły CSS oraz dostosowywać wygląd aplikacji do różnych rozmiarów ekranów.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie arkusza stylów CSS

1. W katalogu projektu utwórz folder o nazwie `styles`, który będzie przeznaczony wyłącznie na pliki CSS.
2. W folderze `styles` utwórz nowy plik o nazwie `main.css`.
3. Otwórz plik `index.html` i zlokalizuj sekcję `<head>`.
4. Dodaj w tej sekcji znacznik `<link>`, który połączy dokument HTML z arkuszem stylów:

```
<link rel="stylesheet" href="styles/main.css">
```

5. Zapisz pliki i upewnij się, że ścieżka do arkusza stylów jest poprawna.

Wynik: arkusz stylów CSS został poprawnie podłączony do dokumentu HTML.

Ćwiczenie 2

Definiowanie podstawowych stylów wizualnych

1. Otwórz plik `main.css`.
2. Dodaj regułę CSS dla selektora `body`, określając:
   - rodzinę czcionek,

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- kolor tła strony,
- podstawowe marginesy i wypełnienia.

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    background-color: #f4f4f4;
}
```

3. Zapisz plik i odśwież stronę w przeglądarce.

Wynik: zmieniony wygląd całej strony – czcionka, tło oraz odstępy.

Ćwiczenie 3

Stylowanie elementów interfejsu użytkownika

1. W pliku `main.css` dodaj reguły stylów dla elementów takich jak `header` i `section`.
2. Zdefiniuj kolory tła, kolory tekstu, odstępy wewnętrzne oraz zewnętrzne.

```
header {
      background-color: #333;
      color: white;
      padding: 16px;
}

section {
      padding: 16px;
      background-color: white;
      margin: 16px;
```


**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
}
```

3. Zapisz plik i odśwież stronę.

Wynik: elementy interfejsu posiadają czytelne rozróżnienie wizualne.

Ćwiczenie 4

Stylowanie przycisków i formularzy

1. W pliku main.css dodaj reguły stylów dla przycisków (button) oraz pól formularzy (input).
2. Określ kolory, obramowania, wypełnienia oraz kursor myszy.

```
button {
      background-color: #007bff;
      color: white;
      border: none;
      padding: 10px 16px;
      cursor: pointer;
}

input {
      padding: 8px;
      margin-bottom: 8px;
}
```

3. Zapisz plik i odśwież stronę w przeglądarce.

Wynik: elementy interaktywne interfejsu są czytelne i spójne wizualnie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 5

Wykorzystanie Flexbox do układu strony

Flexbox (Flexible Box Layout) jest mechanizmem CSS umożliwiającym tworzenie elastycznych układów elementów w jednym wymiarze (w wierszu lub w kolumnie). Pozwala on w prosty sposób kontrolować rozmieszczenie, odstępy oraz kolejność elementów interfejsu, niezależnie od rozmiaru ekranu. Flexbox jest powszechnie stosowany do budowy układów sekcji, paneli, kart oraz nawigacji.

1. W pliku `index.html` w sekcji `<main>` dodaj kilka sekcji aplikacji, np.:

```
<main>
   <section>
        <h2>Sekcja A</h2>
        <p>Treść pierwszej sekcji.</p>
   </section>
   <section>
        <h2>Sekcja B</h2>
        <p>Treść drugiej sekcji.</p>
   </section>
   <section>
        <h2>Sekcja C</h2>
        <p>Treść trzeciej sekcji.</p>
   </section>
</main>
```

2. Następnie w pliku `main.css` ustaw układ Flexbox dla elementu `<main>`:

```
main {
   display: flex;
   gap: 16px;
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
}
```

3. Odśwież stronę w przeglądarce i zaobserwuj rozmieszczenie sekcji.
4. Zmień wartość właściwości flex-direction (np. row, column) i sprawdź wpływ na układ.

Zadania:

1. Sprawdź, jak Flexbox rozmieszcza elementy w jednym wierszu.
2. Zaobserwuj działanie odstępów pomiędzy sekcjami.
3. Przetestuj różne kierunki układu.

Wynik: student rozumie zasadę działania Flexbox i potrafi tworzyć elastyczne układy.

Ćwiczenie 6

dostosowanie aplikacji do różnych ekranów (media queries)

Dodaj regułę media query:

```
@media (max-width: 768px) {
      main {
            flex-direction: column;
      }
}
```

Zmniejsz szerokość okna przeglądarki.

Wynik: układ strony automatycznie dostosowuje się do mniejszych ekranów.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenia samodzielne

Ćwiczenie 1
Zmień schemat kolorów aplikacji.

Ćwiczenie 2
Dodaj efekty `:hover` dla przycisków.

Ćwiczenie 3
Zastosuj Flexbox do stworzenia menu nawigacyjnego.

Ćwiczenie 4
Dodaj kolejne breakpointy dla różnych rozdzielczości.

Ćwiczenie 5
Przetestuj wygląd aplikacji na urządzeniu mobilnym.

Pytania pomocnicze

1. Jaką rolę pełni CSS w aplikacji internetowej?
2. Na czym polega zasada rozdzielenia struktury i prezentacji?
3. Czym jest responsywność interfejsu użytkownika?
4. Jak działają media queries?
5. Dlaczego Flexbox jest użyteczny w projektowaniu UI?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Podsumowanie

W ramach ćwiczenia student zapoznał się z podstawami stylowania interfejsu użytkownika za pomocą CSS oraz z zasadami projektowania wizualnego aplikacji internetowych. Zrealizowane zadania pozwoliły zrozumieć, w jaki sposób style wpływają na wygląd i użyteczność aplikacji oraz jak dostosować jej interfejs do różnych urządzeń. Zdobyta wiedza stanowi podstawę do dalszej pracy z zaawansowanymi technikami stylowania oraz responsywnego projektowania stron WWW.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 5

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Tematem niniejszej instrukcji jest implementacja podstawowych funkcjonalności po stronie klienta, dodawanie dynamicznych elementów interfejsu użytkownika oraz obsługa zdarzeń użytkownika z wykorzystaniem języka JavaScript. Zagadnienia te stanowią kluczowy etap w rozwoju nowoczesnych aplikacji internetowych, umożliwiając przejście od statycznych stron WWW do w pełni interaktywnych aplikacji reagujących na działania użytkownika w czasie rzeczywistym.

JavaScript jest językiem programowania wykonywanym po stronie klienta, który umożliwia manipulację strukturą dokumentu HTML za pośrednictwem obiektu DOM (Document Object Model). DOM reprezentuje dokument HTML w postaci drzewa obiektów, dzięki czemu możliwy jest dynamiczny dostęp do elementów strony, ich modyfikacja, usuwanie oraz tworzenie nowych elementów w trakcie działania aplikacji. Mechanizm ten pozwala na aktualizowanie zawartości strony bez konieczności jej ponownego ładowania.

Jednym z fundamentalnych paradygmatów wykorzystywanych w aplikacjach webowych jest programowanie zdarzeniowe. Polega ono na reagowaniu aplikacji na zdarzenia generowane przez użytkownika lub środowisko przeglądarki, takie jak kliknięcia myszy, naciśnięcia klawiszy, zmiany wartości pól formularzy czy przesyłanie danych. W JavaScript obsługa zdarzeń realizowana jest poprzez rejestrowanie tzw. nasłuchiwaczy zdarzeń (event listeners), które przypisują określone funkcje do konkretnych zdarzeń.

Dynamiczne elementy interfejsu użytkownika odgrywają istotną rolę w poprawie użyteczności i ergonomii aplikacji. Umożliwiają one m.in. wyświetlanie komunikatów, walidację danych wprowadzanych przez użytkownika, aktualizację treści strony, a także budowę interaktywnych komponentów, takich jak listy, formularze czy panele informacyjne. Umiejętność tworzenia takich elementów jest niezbędna w pracy frontendowego programisty.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

W ramach niniejszego laboratorium student zapozna się z podstawowymi mechanizmami JavaScript wykorzystywanymi do obsługi zdarzeń oraz dynamicznej modyfikacji interfejsu użytkownika. Zdobyta wiedza stanowi fundament do dalszej nauki zaawansowanych technik programowania frontendowego oraz pracy z nowoczesnymi bibliotekami i frameworkami JavaScript.

Cel laboratorium

Celem laboratorium jest nauczenie studenta implementacji podstawowych funkcjonalności aplikacji po stronie klienta z wykorzystaniem JavaScript. Student pozna zasady obsługi zdarzeń użytkownika, manipulacji elementami DOM oraz tworzenia dynamicznych elementów interfejsu.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie plików projektu

1. W katalogu projektu utwórz folder scripts.
2. W folderze scripts utwórz plik main.js.
3. W pliku index.html dodaj w sekcji <head> lub przed zamknięciem znacznika <body> odwołanie do pliku JavaScript:

```
<script src="scripts/main.js"></script>
```

Wynik: projekt jest przygotowany do wykonywania skryptów JavaScript po stronie klienta.

Ćwiczenie 2

Reagowanie na zdarzenie kliknięcia

1. W pliku index.html dodaj przycisk:

```
<button id="btn">Kliknij mnie</button>
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

2. W pliku main.js pobierz przycisk i zarejestruj obsługę zdarzenia click:

```
const button = document.getElementById('btn');


button.addEventListener('click', () => {
   alert('Przycisk został kliknięty');
});
```

3. Odśwież stronę i kliknij przycisk.

Wynik: aplikacja reaguje na interakcję użytkownika poprzez obsługę zdarzenia kliknięcia.

Ćwiczenie 3

Dynamiczna zmiana treści strony

1. W dokumencie HTML dodaj element tekstowy:

```
<p id="text">Tekst początkowy</p>
```

2. Zmodyfikuj kod JavaScript, aby po kliknięciu przycisku zmieniać treść akapitu:

```
const text = document.getElementById('text');


button.addEventListener('click', () => {
   text.textContent = 'Tekst został zmieniony dynamicznie';
});
```

Wynik: student potrafi dynamicznie modyfikować treść elementów DOM.

Ćwiczenie 4

Dodawanie elementów interfejsu w czasie działania

1. W pliku HTML dodaj pusty kontener:

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
<div id="container"></div>
```

2. W pliku `main.js` dodaj kod tworzący nowy element po kliknięciu przycisku:

```
const container = document.getElementById('container');


button.addEventListener('click', () => {
   const newElement = document.createElement('p');
   newElement.textContent = 'Nowy element interfejsu';
   container.appendChild(newElement);
});
```

Wynik: student potrafi dynamicznie tworzyć i dodawać elementy interfejsu użytkownika.

Ćwiczenie 5

Obsługa zdarzeń formularza

1. Dodaj do HTML prosty formularz:

```
<form id="form">
   <input type="text" id="name" placeholder="Imię">
   <button type="submit">Wyślij</button>
</form>
```

2. W JavaScript obsłuż zdarzenie `submit`:

```
const form = document.getElementById('form');


form.addEventListener('submit', (event) => {
   event.preventDefault();
   alert('Formularz został wysłany');
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
});
```

Wynik: student rozumie obsługę zdarzeń formularza oraz mechanizm zapobiegania domyślnemu działaniu przeglądarki.

Ćwiczenie 6

Obsługa zdarzeń klawiatury (keydown) i reakcja interfejsu

1. W pliku `index.html` dodaj element do wyświetlania informacji:

```
<p id="keyInfo">Naciśnij dowolny klawisz…</p>
```

2. W pliku `main.js` pobierz element i dodaj nasłuchiwanie zdarzenia `keydown` na obiekcie `window`:

```
const keyInfo = document.getElementById('keyInfo');


window.addEventListener('keydown', (event) => {
   keyInfo.textContent = `Wciśnięto klawisz: ${event.key}`;
});
```

3. Odśwież stronę i naciśnij kilka różnych klawiszy (np. litery, Enter, strzałki).

Wynik: student potrafi obsłużyć zdarzenia klawiatury oraz wykorzystać dane z obiektu zdarzenia (`event.key`) do aktualizacji interfejsu.

Ćwiczenie 7

Wykorzystanie obiektu zdarzenia i elementu docelowego (event.target)

1. W pliku `index.html` dodaj trzy przyciski w kontenerze:

```
<div id="actions">
   <button class="action">Akcja 1</button>
   <button class="action">Akcja 2</button>
   <button class="action">Akcja 3</button>
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
  </div>
  <p id="actionResult">Wybierz akcję…</p>
```

2. W pliku `main.js` pobierz wszystkie przyciski i dla każdego dodaj nasłuchiwanie `click`, korzystając z `event.target`:

```
const actionResult = document.getElementById('actionResult');
const actionButtons = document.querySelectorAll('.action');


actionButtons.forEach((btn) => {
   btn.addEventListener('click', (event) => {
        const clicked = event.target;
        actionResult.textContent = `Wybrano: ${clicked.textContent}`;
   });
});
```

3. Odśwież stronę i klikaj różne przyciski.

Wynik: student rozumie rolę obiektu zdarzenia, potrafi identyfikować element wywołujący zdarzenie (`event.target`) oraz dynamicznie aktualizować treść interfejsu.

Ćwiczenie 8

Walidacja formularza po stronie klienta

Celem tego zadania jest sprawdzenie poprawności danych wprowadzanych przez użytkownika przed wysłaniem formularza oraz wyświetlenie odpowiednich komunikatów w interfejsie aplikacji.

1. Rozszerz formularz w pliku `index.html`, dodając miejsce na komunikat walidacyjny:

```
<form id="form">
```


Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
    <input type="text" id="name" placeholder="Imię">
    <span id="error" style="color: red;"></span><br>
    <button type="submit">Wyślij</button>
</form>
```

2. W pliku `main.js` zmodyfikuj obsługę zdarzenia `submit`, dodając walidację pola tekstowego:

```
const error = document.getElementById('error');

form.addEventListener('submit', (event) => {
    event.preventDefault();

    if (name.value.trim() === '') {
        error.textContent = 'Pole imię nie może być puste';
    } else {
        error.textContent = '';
        alert(`Wysłano formularz. Imię: ${name.value}`);
    }
});
```

3. Przetestuj działanie formularza, próbując wysłać go z pustym oraz wypełnionym polem.

Zadania:

1. Sprawdź, czy formularz nie jest wysyłany, gdy pole jest puste.
2. Zaobserwuj sposób wyświetlania komunikatu błędu.

Wynik: student potrafi zaimplementować podstawową walidację formularza po stronie klienta, zapobiec wysłaniu niepoprawnych danych oraz wyświetlić komunikaty walidacyjne w interfejsie użytkownika.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenia samodzielne

Ćwiczenie 1
Dodaj drugi przycisk, który usuwa ostatnio dodany element.

Ćwiczenie 2
Zaimplementuj licznik kliknięć przycisku.

Ćwiczenie 3
Zmień kolor tekstu po najechaniu kursorem myszy.

Ćwiczenie 4
Wyświetl wartość wpisaną do pola formularza na stronie.

Ćwiczenie 5
Dodaj obsługę zdarzenia `keydown` dla klawiatury.

Ćwiczenie 6
Dodaj walidację formularza: jeśli pole „Imię" jest puste, wyświetl komunikat na stronie zamiast wysyłać formularz.

Ćwiczenie 7
Zaimplementuj prosty „tryb ciemny": po kliknięciu przycisku przełącz klasę na `body` i zmień wygląd strony (np. tło/kolor tekstu).

Pytania pomocnicze

1. Czym jest programowanie zdarzeniowe?
2. Jaką rolę pełni obiekt DOM w JavaScript?
3. Jaka jest różnica pomiędzy `addEventListener` a atrybutami zdarzeń HTML?
4. Dlaczego stosuje się `event.preventDefault()`?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

5. Jakie są zalety dynamicznego tworzenia elementów interfejsu?

Podsumowanie

W ramach ćwiczenia student zaimplementował podstawowe funkcjonalności po stronie klienta z wykorzystaniem JavaScript oraz nauczył się obsługi zdarzeń użytkownika. Zrealizowane zadania pozwoliły zrozumieć mechanizmy dynamicznej modyfikacji interfejsu oraz reagowania aplikacji na działania użytkownika. Zdobyta wiedza ma bezpośrednie zastosowanie w tworzeniu interaktywnych aplikacji webowych i stanowi fundament dalszej nauki programowania frontendowego.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 6

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Współczesne aplikacje webowe bardzo rzadko działają jako jednorodne, monolityczne systemy. Zamiast tego są one najczęściej budowane w oparciu o architekturę klient–serwer, w której warstwa kliencka (front-end) odpowiada za interfejs użytkownika, natomiast warstwa serwerowa (back-end) realizuje logikę biznesową, przetwarzanie danych oraz komunikację z bazami danych. Kluczowym elementem łączącym te dwie warstwy jest API (Application Programming Interface), czyli zestaw jasno zdefiniowanych reguł umożliwiających wymianę danych pomiędzy różnymi komponentami systemu.

W kontekście aplikacji webowych najczęściej spotykanym podejściem jest tworzenie tzw. API HTTP, które udostępnia określone zasoby za pomocą punktów końcowych (endpointów) i standardowych metod protokołu HTTP, takich jak GET, POST, PUT czy DELETE. Takie API często przyjmuje postać REST API, gdzie dane są przesyłane w lekkim formacie (najczęściej JSON), a każda operacja ma jasno określone znaczenie semantyczne.

Przygotowanie podstawowego API po stronie serwera wymaga nie tylko zaprogramowania logiki obsługi żądań, ale również właściwej konfiguracji środowiska serwerowego. Obejmuje to uruchomienie serwera aplikacji, obsługę połączeń HTTP, parsowanie danych przychodzących od klienta oraz generowanie odpowiedzi w odpowiednim formacie. Istotnym aspektem jest także zrozumienie mechanizmu komunikacji asynchronicznej pomiędzy klientem i serwerem, który pozwala na dynamiczne pobieranie i wysyłanie danych bez konieczności przeładowywania całej strony.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Cel laboratorium

Celem laboratorium jest zapoznanie studentów z procesem tworzenia prostego API po stronie serwera, konfiguracją serwera aplikacji oraz definiowaniem punktów końcowych obsługujących podstawowe operacje HTTP. Student nauczy się, w jaki sposób serwer odbiera żądania od klienta, przetwarza dane i zwraca odpowiedzi w formacie JSON. Ćwiczenie ma na celu zbudowanie podstawowego zrozumienia komunikacji klient–serwer.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie środowiska serwerowego.

Utworzenie struktury projektu.

1. Utwórz nowy katalog projektu, który będzie pełnił rolę katalogu głównego aplikacji serwerowej (np. `api-server`).
2. W katalogu projektu utwórz plik startowy serwera, np. `server.js` lub `index.js`.

Utworzenie podstawowego serwera http.

W pliku startowym należy utworzyć serwer HTTP i wskazać port, na którym będzie on nasłuchiwał. Przykładowo, w środowisku Node.js oznacza to:

- zaimportowanie modułu umożliwiającego obsługę serwera HTTP,
- zdefiniowanie funkcji obsługującej żądania (request) i odpowiedzi (response),
- uruchomienie serwera na wybranym porcie (np. 3000).

Na tym etapie serwer nie musi jeszcze obsługiwać żadnej logiki biznesowej – jego zadaniem jest jedynie poprawne uruchomienie i reagowanie na połączenia.

Konfiguracja portu i uruchomienie serwera.

1. W kodzie serwera ustaw numer portu, na którym aplikacja będzie dostępna lokalnie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

2. Uruchom serwer z poziomu terminala (np. poleceniem uruchamiającym plik startowy).
3. Sprawdź w konsoli, czy pojawiła się informacja potwierdzająca start serwera.

Weryfikacja działania serwera.

1. Otwórz przeglądarkę internetową.
2. Wpisz adres w postaci http://localhost:PORT, gdzie PORT to numer portu ustawiony w aplikacji.
3. Upewnij się, że serwer odpowiada (np. zwracając prosty komunikat lub pustą odpowiedź bez błędu).

Wynik: Serwer aplikacji jest poprawnie skonfigurowany, uruchomiony lokalnie i nasłuchuje na określonym porcie, dzięki czemu może przyjmować żądania HTTP od klienta.

Ćwiczenie 2

Obsługa podstawowego żądania HTTP.

Zaprojektowanie ścieżki endpointu.

1. Wybierz czytelną ścieżkę zgodną z konwencją API (np. `/api/status`, `/api/health`, `/api/ping`).
2. Ustal, że endpoint ma służyć wyłącznie do odczytu informacji o stanie serwera (dlatego użyjesz metody GET).

Konfigurowanie obsługi metody GET.

1. W pliku serwera dodaj regułę routingu, która wykryje:
   - że przychodzące żądanie ma metodę GET,
   - że jego adres (ścieżka) odpowiada `/api/status`.
2. Po spełnieniu warunków wywołaj funkcję obsługi (handler), w której zbudujesz odpowiedź.

W zależności od wybranego podejścia technicznego może to wyglądać jako:

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- warunek sprawdzający `method` i `url` (w prostym serwerze HTTP), albo
- definicja trasy w frameworku serwerowym (routing).

Przygotowanie odpowiedzi serwera.

W odpowiedzi serwer powinien zwrócić informację potwierdzającą działanie API. Zalecane jest użycie formatu JSON.

1. Ustal treść odpowiedzi, np. obiekt zawierający pola:
   - `status` (np. `"ok"`),
   - `message` (np. `"API działa poprawnie"`),
   - opcjonalnie `timestamp` (czas odpowiedzi) lub `version`.
2. Ustaw kod statusu HTTP:
   - 200 dla poprawnej odpowiedzi.
3. Ustaw nagłówek `Content-Type` na `application/json` (gdy zwracasz JSON).
4. Zwróć odpowiedź do klienta.

Krok 4. Testowanie endpointu.

Test powinien potwierdzić trzy elementy: (1) poprawną ścieżkę, (2) poprawną metodę HTTP, (3) poprawny format odpowiedzi.

1. Test w przeglądarce:
   - Wejdź na adres `http://localhost:PORT/api/status`.
   - Sprawdź, czy wyświetla się odpowiedź (JSON lub tekst).
2. Test narzędziem developerskim (DevTools):
   - Otwórz zakładkę Network.
   - Odśwież stronę lub wywołaj endpoint.
   - Sprawdź kod statusu (powinien być 200) oraz nagłówki (Content-Type).
3. Test narzędziem do API (np. Postman/Insomnia) lub wierszem poleceń:
   - Wyślij żądanie GET na `http://localhost:PORT/api/status`.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- Porównaj odpowiedź z założonym formatem.

Warto również wykonać test negatywny:

- wywołać nieistniejącą ścieżkę (np. `/api/statu`), aby upewnić się, że serwer zwraca błąd (np. 404),
- wywołać tę samą ścieżkę inną metodą (np. POST), aby sprawdzić reakcję serwera.

Wynik: Po wywołaniu endpointu `GET /api/status` klient otrzymuje odpowiedź z kodem 200 oraz treścią w ustalonym formacie (najczęściej JSON). Testy w przeglądarce lub narzędziu do API potwierdzają, że serwer poprawnie rozpoznaje metodę GET, ścieżkę endpointu i zwraca właściwe nagłówki.

Ćwiczenie 3

Praca z danymi w formacie JSON.

1. Przygotuj po stronie serwera przykładową strukturę danych (np. tablicę obiektów).
2. Zdefiniuj endpoint typu GET (np. `/api/items`), który zwraca całą kolekcję danych.
3. Skonfiguruj nagłówki odpowiedzi tak, aby jednoznacznie wskazywały na format JSON.
4. Sprawdź poprawność struktury zwracanych danych.

Wynik: Klient otrzymuje odpowiedź w postaci poprawnie sformatowanego obiektu JSON zawierającego listę danych.

Ćwiczenie 4

Obsługa żądań POST – dodawanie danych.

1. Zdefiniuj endpoint obsługujący metodę POST (np. `/api/items`).
2. Skonfiguruj mechanizm odczytu danych przesyłanych w treści żądania.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

3. Dodaj logikę umożliwiającą zapis nowych danych w strukturze przechowywanej po stronie serwera.
4. Przygotuj odpowiedź potwierdzającą wykonanie operacji.

Wynik: Serwer poprawnie odbiera dane przesłane w żądaniu POST, zapisuje je i zwraca komunikat potwierdzający dodanie nowego zasobu.

Ćwiczenie 5

Obsługa parametrów i identyfikatorów zasobów.

W tym etapie student uczy się obsługi dynamicznych parametrów w ścieżkach URL.

1. Zdefiniuj endpoint z parametrem w ścieżce (np. `/api/items/:id`).
2. Obsłuż metodę GET dla tego endpointu.
3. Odczytaj przekazany identyfikator i wykorzystaj go do wyszukania odpowiedniego elementu.
4. Zwróć znaleziony obiekt lub informację o jego braku.

Wynik: Serwer zwraca dane pojedynczego zasobu wskazanego przez identyfikator lub komunikat o błędzie, jeśli zasób nie istnieje.

Ćwiczenie 6

Obsługa błędów i kodów statusu http.

Zidentyfikuj sytuacje błędne w istniejących endpointach.

Na podstawie poprzednich ćwiczeń (2–5) wypisz, gdzie mogą wystąpić błędy. Przykładowo:

- GET /api/items – kolekcja jest pusta lub nie została poprawnie zainicjalizowana.
- GET /api/items/:id – brak zasobu o podanym `id`.
- POST /api/items – brak danych w treści żądania, brak wymaganych pól, nieprawidłne typy wartości.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- Dowolny endpoint – nieobsługiwana metoda HTTP lub błędna ścieżka.

Zaimplementuj wspólny schemat odpowiedzi błędu.

Aby ułatwić klientowi przetwarzanie błędów, przyjmij jeden, stały format odpowiedzi (JSON). Zalecana struktura:

- `error`: krótki typ błędu (np. `"NotFound"`, `"ValidationError"`),
- `message`: opis błędu zrozumiały dla człowieka,
- opcjonalnie `details`: lista szczegółów (np. które pola są błędne).

Ważne: nawet w przypadku błędu serwer powinien zwrócić czytelną odpowiedź oraz prawidłowe nagłówki (np. `Content-Type: application/json`).

Implementacja obsługi błędów w endpointach.

W każdym endpointcie zastosuj poniższą sekwencję:

1. Walidacja danych wejściowych
   - Dla endpointów z parametrem `:id` sprawdź, czy parametr istnieje i czy ma oczekiwany format (np. liczba całkowita).
   - Dla endpointów POST sprawdź, czy treść żądania została w ogóle dostarczona oraz czy zawiera wymagane pola.
2. Obsługa wyniku walidacji
   - Jeśli walidacja się nie powiodła, zakończ obsługę żądania natychmiast i zwróć:
     - kod 400 Bad Request,
     - obiekt JSON z informacją o błędzie i (opcjonalnie) listą brakujących/błędnych pól.
3. Wyszukiwanie zasobu (gdy dotyczy)
   - Jeżeli endpoint operuje na zasobie (np. `/api/items/:id`), wyszukaj element w kolekcji.
4. Obsługa braku zasobu
   - Jeśli zasób nie istnieje, zwróć:

- kod 404 Not Found,
- obiekt JSON z komunikatem, że element o podanym identyfikatorze nie został znaleziony.

5. Obsługa „nieobsługiwanej metody" i „nieznanej ścieżki"
   - Jeżeli serwer otrzyma żądanie na ścieżkę, której nie obsługujesz, zwróć:
     - kod 404 Not Found.
   - Jeżeli ścieżka jest poprawna, ale metoda nieobsługiwana (np. POST na endpoint tylko do odczytu), zwróć:
     - kod 405 Method Not Allowed (jeśli takie rozróżnienie jest realizowane w Twojej implementacji).

Dodaj obsługę błędów nieprzewidzianych.

Poza błędami walidacji i brakiem zasobu mogą wystąpić błędy wykonania (np. błąd parsowania danych, wyjątek w logice). Dla takich sytuacji:

1. Zadbaj o przechwycenie błędu (mechanizm zależny od technologii: blok try/catch lub centralny handler błędów).
2. Zwróć:
   - kod 500 Internal Server Error,
   - komunikat ogólny (bez ujawniania szczegółów technicznych w odpowiedzi).
3. Zapisz szczegóły błędu w logach serwera (np. w konsoli), aby ułatwić diagnostykę.

Przetestuj przypadki poprawne i błędne.

Dla każdego endpointu wykonaj testy:

- przypadek poprawny (np. istniejące id, kompletne dane),
- przypadek błędny:
  - błędny format danych (400),
  - nieistniejący zasób (404),

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- niepoprawna ścieżka (404),
- nieobsługiwana metoda (405 – jeśli wdrożone),
- błąd awaryjny (500 – zasymulowany).

Wynik: API zwraca jednoznaczne odpowiedzi zawierające właściwe kody statusu HTTP (np. 200/201, 400, 404, 405, 500) oraz spójne komunikaty w formacie JSON. Dzięki temu klient może poprawnie interpretować wynik operacji, a developer łatwiej diagnozuje problemy na podstawie logów i treści odpowiedzi.

Ćwiczenia samodzielne

Ćwiczenie 1
Rozszerz API o endpoint umożliwiający usuwanie danych (metoda DELETE).

Ćwiczenie 2
Dodaj obsługę aktualizacji istniejącego zasobu (metoda PUT lub PATCH).

Ćwiczenie 3
Zaimplementuj prostą walidację danych przesyłanych w żądaniu POST.

Ćwiczenie 4
Dodaj dodatkowy endpoint zwracający dane w zależności od parametrów zapytania.

Ćwiczenie 5
Przygotuj dokumentację API w formie opisu endpointów i przykładów żądań.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Pytania pomocnicze

1. Czym jest API i jaką rolę pełni w architekturze klient–serwer?
2. Jakie znaczenie mają metody HTTP w kontekście REST API?
3. Dlaczego format JSON jest powszechnie stosowany w komunikacji klient–serwer?
4. Jakie są różnice pomiędzy metodami GET i POST?
5. Dlaczego ważne jest stosowanie odpowiednich kodów statusu HTTP?
6. W jaki sposób serwer rozpoznaje, jaką operację należy wykonać dla danego endpointu?

Podsumowanie

W ramach ćwiczenia zaprojektowano i uruchomiono podstawowe API po stronie serwera, umożliwiające obsługę prostych operacji na danych. Zrealizowane zadania pozwoliły zrozumieć mechanizm komunikacji klient–serwer oraz rolę punktów końcowych i metod HTTP. Student zdobył praktyczne umiejętności związane z konfiguracją serwera aplikacji i przetwarzaniem żądań. Warto zastanowić się, jakie elementy należałoby dodać, aby API było bezpieczne i gotowe do zastosowań produkcyjnych. Wiedza zdobyta podczas ćwiczenia stanowi podstawę do dalszej pracy nad bardziej złożonymi systemami webowymi.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 7

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Formularze stanowią jeden z podstawowych mechanizmów interakcji użytkownika z aplikacją internetową. Umożliwiają one wprowadzanie danych, które następnie mogą być przetwarzane po stronie klienta lub przesyłane do serwera w celu dalszej obsługi. W klasycznym modelu działania aplikacji webowych formularz pełni rolę interfejsu pomiędzy użytkownikiem a logiką biznesową systemu.

Dane wprowadzane w formularzach wymagają weryfikacji, aby zapewnić ich poprawność, kompletność oraz bezpieczeństwo. Walidacja może być realizowana na dwóch poziomach. Walidacja po stronie klienta (najczęściej z wykorzystaniem HTML5 i JavaScript) pozwala szybko wykryć błędy jeszcze przed wysłaniem danych na serwer, poprawiając komfort użytkownika i ograniczając liczbę niepotrzebnych żądań HTTP. Przykładami takich mechanizmów są atrybuty required, type, min, max oraz wykorzystanie zdarzeń formularza w JavaScript.

Z kolei walidacja po stronie serwera jest niezbędna niezależnie od walidacji klienckiej, ponieważ dane wysyłane przez użytkownika mogą zostać zmodyfikowane lub przesłane z pominięciem interfejsu użytkownika. Serwer odpowiada za ostateczne sprawdzenie poprawności danych, ich przetworzenie oraz zapis lub wykorzystanie w dalszych operacjach.

Nowoczesne aplikacje internetowe często wykorzystują asynchroniczne przesyłanie danych formularzy (np. przy użyciu Fetch API), co pozwala na komunikację z serwerem bez przeładowywania strony. Dzięki temu możliwe jest tworzenie bardziej dynamicznych i responsywnych interfejsów użytkownika.

W ramach niniejszego laboratorium student zapozna się z obsługą formularzy w JavaScript, walidacją danych po stronie klienta oraz przesyłaniem danych formularza do serwera API i ich podstawowym przetwarzaniem.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest nabycie umiejętności obsługi formularzy w aplikacji webowej, walidacji danych po stronie klienta oraz przesyłania danych do serwera. Student nauczy się reagować na zdarzenia formularza oraz analizować odpowiedzi serwera.

Instrukcja laboratoryjna

Ćwiczenie 1

Utworzenie formularza HTML i zaplanowanie pól danych.

1. Utwórz plik `index.html` (lub otwórz istniejący).
2. Wewnątrz `<body>` dodaj formularz o identyfikatorze `registerForm`.
3. Dodaj pola:
   - Imię jako `type="text"` (wymagane),
   - E-mail jako `type="email"` (wymagane).
4. Dodaj przycisk typu `submit`.
5. Dodaj element na komunikaty dla użytkownika (np. `<p id="message"></p>`), aby wyświetlać wyniki walidacji lub odpowiedzi serwera.

```
<form id="registerForm">
<label>
Imię:
<input type="text" id="name" required>
</label><br>
<label>
E-mail:
<input type="email" id="email" required>
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
</label><br>
<button type="submit">Wyślij</button>
</form>
<p id="message"></p>
```

Wynik: formularz jest widoczny w przeglądarce; atrybuty `required` i `type="email"` uruchamiają podstawową walidację HTML5.

Ćwiczenie 2

Podłączenie JavaScript i przechwycenie wysyłki formularza (submit).

1. Utwórz plik `main.js`.
2. Podłącz go w `index.html` (najprościej przed `</body>`, żeby elementy HTML były już wczytane):

```
<script src="main.js"></script>
```

3. W `main.js` pobierz formularz i pole na komunikaty.
4. Dodaj obsługę zdarzenia `submit`, użyj `event.preventDefault()`.
5. Odczytaj wartości pól i wyświetl je w `#message` (na razie bez wysyłania na serwer).

```
const form = document.getElementById('registerForm');
const message = document.getElementById('message');

form.addEventListener('submit', (event) => {
   event.preventDefault();

   const name = document.getElementById('name').value;
   const email = document.getElementById('email').value;
```


**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    message.textContent = `Wprowadzono: ${name}, ${email}`;
});
```

Wynik: po kliknięciu „Wyślij” strona nie przeładowuje się, a dane są odczytywane i wyświetlane w interfejsie.

Ćwiczenie 3

Walidacja danych po stronie klienta w JavaScript (logika + komunikaty).

1. Rozszerz obsługę `submit` o walidację pola imię:
   - usuń białe znaki (`trim()`),
   - sprawdź minimalną długość (np. 3 znaki).
2. W przypadku błędu wyświetl komunikat i przerwij dalsze wykonywanie (`return`).
3. Jeśli dane są poprawne, wyświetl komunikat „Dane poprawne.”

```
form.addEventListener('submit', (event) => {
    event.preventDefault();

    const name = document.getElementById('name').value.trim();
    const email = document.getElementById('email').value.trim();

    if (name.length < 3) {
        message.textContent = 'Imię musi zawierać co najmniej 3
        znaki.';
        return;
    }

    // Przykładowa dodatkowa kontrola (opcjonalnie):
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    if (!email.includes('@')) {
        message.textContent = 'Podaj poprawny adres e-mail.';
        return;
    }

    message.textContent = 'Dane poprawne.';
});
```

Wynik: formularz nie przechodzi dalej, gdy dane są niepoprawne; użytkownik dostaje jasny komunikat.

Ćwiczenie 4

Wysyłanie danych formularza do serwera metodą POST (Fetch API)

1. Załóż, że serwer udostępnia endpoint `POST http://localhost:3000/register`.
2. Po udanej walidacji zbuduj obiekt danych i wyślij go metodą `POST`.
3. Ustaw nagłówek `Content-Type: application/json`.
4. Odbierz odpowiedź jako JSON i wyświetl komunikat użytkownikowi.

```
form.addEventListener('submit', async (event) => {
    event.preventDefault();

    const name = document.getElementById('name').value.trim();
    const email = document.getElementById('email').value.trim();

    if (name.length < 3) {
        message.textContent = 'Imię musi zawierać co najmniej 3
        znaki.';
```


**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
            return;
        }

        const data = { name, email };

        try {
            const response = await fetch(
            'http://localhost:3000/register', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(data)
            });

            const result = await response.json();

            if (!response.ok) {
                message.textContent = `Błąd: ${result.message || 'Nie
                udało się wysłać danych'}`;
                return;
            }

            message.textContent = result.message;
        } catch (err) {
            message.textContent = 'Błąd połączenia z serwerem.';
        }
    });
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wynik: formularz wysyła dane jako JSON do serwera bez przeładowania strony, a aplikacja wyświetla odpowiedź serwera.

Ćwiczenie 5

Przetwarzanie danych formularza po stronie serwera (Express) i walidacja serwerowa.

1. Upewnij się, że serwer Express ma włączony parser JSON:

```
app.use(express.json());
```

2. Dodaj endpoint `POST /register`, który:
   - odczyta `name` i `email` z `req.body`,
   - sprawdzi, czy pola istnieją,
   - zwróci błąd 400, jeśli dane są niepoprawne,
   - zwróci 200, jeśli „rejestracja" zakończyła się sukcesem.

```
app.post('/register', (req, res) => {
   const { name, email } = req.body;

   if (!name || !email) {
        return res.status(400).json({ message: 'Brak wymaganych
        danych (name, email).' });
   }

   if (String(name).trim().length < 3) {
        return res.status(400).json({ message: 'Imię musi mieć min.
        3 znaki.' });
   }
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    // Tu w realnej aplikacji byłby zapis do bazy / logika biznesowa
    res.status(200).json({ message: 'Rejestracja zakończona
    sukcesem.' });
});
```

Wynik: serwer poprawnie odbiera dane, wykonuje walidację serwerową i zwraca odpowiedź JSON z odpowiednim kodem statusu.

Ćwiczenie 6

Interpretacja odpowiedzi serwera i obsługa błędów w interfejsie.

1. W kodzie Fetch (Ćwiczenie 4) zwróć uwagę na `response.ok`.
2. Przetestuj dwa scenariusze:
   - poprawne dane (np. imię ≥ 3 znaki),
   - błędne dane (np. puste pole lub zbyt krótkie imię).
3. Sprawdź, czy komunikat w `#message` zmienia się adekwatnie.

Wynik: interfejs użytkownika poprawnie rozróżnia sukces i błąd oraz wyświetla komunikaty zwrócone przez serwer.

Ćwiczenie 7

Wysyłanie formularza metodą PUT (aktualizacja danych).

W ćwiczeniu przyjmiemy, że aktualizujemy dane użytkownika o `id = 1` poprzez endpoint: PUT `http://localhost:3000/users/1`.

1. Dodaj endpoint PUT po stronie serwera

```
app.put('/users/:id', (req, res) => {
    const id = Number(req.params.id);
    const { name, email } = req.body;
```


Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
    if (!name || !email) {\
        return res.status(400).json({ message: 'Brak wymaganych
        danych (name, email).' });
    }

    // W realnej aplikacji: aktualizacja w bazie dla id
    return res.status(200).json({ message: `Zaktualizowano
    użytkownika ${id}.`, data: { id, name, email } });
});
```

2. Wyślij dane formularza metodą PUT po stronie klienta

2.1. Dodaj drugi przycisk w HTML, który będzie służył do aktualizacji danych (lub zrób osobny formularz). Najprościej:

```
<button type="button" id="updateBtn">Aktualizuj (PUT)</button>
```

2.2. W `main.js` dodaj obsługę kliknięcia i wyślij żądanie `PUT`:

```
const updateBtn = document.getElementById('updateBtn');

updateBtn.addEventListener('click', async () => {
    const name = document.getElementById('name').value.trim();
    const email = document.getElementById('email').value.trim();

    if (name.length < 3) {
        message.textContent = 'Imię musi zawierać co najmniej 3
        znaki.';
        return;
    }
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    try {
        const response = await fetch(
        'http://localhost:3000/users/1', {
            method: 'PUT',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ name, email })
        });

        const result = await response.json();

        if (!response.ok) {
            message.textContent = `Błąd: ${result.message ||
            'Aktualizacja nieudana'}`;
            return;
        }

        message.textContent = result.message;
    } catch {
        message.textContent = 'Błąd połączenia z serwerem.';
    }
});
```

Wynik: student potrafi wysłać dane formularza metodą PUT w celu aktualizacji zasobu oraz obsłużyć odpowiedź serwera w interfejsie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenia samodzielne

Ćwiczenie 1
Dodaj pole „hasło” i sprawdź minimalną długość hasła po stronie klienta.

Ćwiczenie 2
Zaimplementuj potwierdzenie hasła (dwa pola muszą zawierać tę samą wartość).

Ćwiczenie 3
Dodaj walidację adresu e-mail z użyciem wyrażenia regularnego.

Ćwiczenie 4
Wyświetl komunikaty błędów pod konkretnymi polami formularza.

Ćwiczenie 5
Zaimplementuj wysyłanie formularza metodą PUT.

Ćwiczenie 6
Dodaj reset formularza po poprawnym wysłaniu danych.

Pytania pomocnicze

1. Dlaczego walidacja po stronie klienta nie wystarcza w aplikacjach webowych?
2. Jakie zalety ma asynchroniczne przesyłanie danych formularzy?
3. Czym różni się walidacja HTML5 od walidacji realizowanej w JavaScript?
4. Jaką rolę pełni event.preventDefault() w obsłudze formularzy?
5. Dlaczego serwer zawsze powinien ponownie walidować dane?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Podsumowanie

W ramach ćwiczenia student zapoznał się z pełnym procesem obsługi formularzy w aplikacji webowej – od zaprojektowania formularza HTML, poprzez walidację danych po stronie klienta, aż po asynchroniczne przesyłanie danych do serwera i ich przetwarzanie. Zrealizowane zadania pozwoliły zrozumieć znaczenie przechwytywania zdarzeń formularza, zapobiegania domyślnemu zachowaniu przeglądarki oraz interpretowania odpowiedzi serwera. Ćwiczenie podkreśliło konieczność stosowania walidacji zarówno po stronie klienta, jak i serwera, a także różnice pomiędzy wykorzystaniem metod POST i PUT w kontekście tworzenia oraz aktualizacji danych. Zdobyte umiejętności mają bezpośrednie zastosowanie w tworzeniu nowoczesnych, bezpiecznych i interaktywnych aplikacji internetowych oraz stanowią solidną podstawę do dalszej pracy z formularzami i komunikacją frontend–backend.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 8

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

W większości aplikacji webowych dane muszą być przechowywane w sposób trwały, spójny i możliwy do współdzielenia między użytkownikami. W tym celu wykorzystuje się systemy zarządzania bazami danych (DBMS), które odpowiadają za przechowywanie rekordów, zapewnienie integralności danych, kontrolę dostępu oraz wydajne wykonywanie zapytań. Jednym z najczęściej spotykanych modeli przechowywania danych jest model relacyjny, w którym informacje są organizowane w tabele (z wierszami i kolumnami), a powiązania między danymi opisuje się za pomocą kluczy głównych i obcych.

W praktyce pracy z bazą danych bardzo często wraca zestaw czterech podstawowych operacji określanych skrótem CRUD (Create, Read, Update, Delete). Operacje te odpowiadają kolejno za tworzenie nowych rekordów, odczyt danych (pojedynczego rekordu lub listy), aktualizację istniejących informacji oraz usuwanie danych. Model CRUD stanowi fundament budowy systemów informatycznych: od prostych list zadań po rozbudowane systemy biznesowe.

W niniejszym laboratorium jako relacyjny silnik bazy danych wykorzystasz MariaDB, czyli wydajny i popularny system kompatybilny z MySQL. Aplikacja serwerowa będzie łączyła się z bazą danych za pomocą danych uwierzytelniających (host, użytkownik, hasło, nazwa bazy), a następnie będzie wykonywała zapytania SQL realizujące CRUD. Istotną częścią pracy jest rozumienie różnicy między warstwą aplikacji a bazą danych: aplikacja odpowiada za walidację danych wejściowych, logikę biznesową i format odpowiedzi (np. JSON), natomiast baza danych odpowiada za trwałość, spójność i wykonywanie zapytań.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest skonfigurowanie bazy MariaDB oraz utworzenie przykładowej tabeli do przechowywania danych. Następnie student łączy aplikację serwerową z bazą i implementuje operacje CRUD jako zestaw endpointów API. Po zakończeniu ćwiczenia student potrafi przetestować operacje SQL oraz potwierdzić zmiany w bazie danych.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie bazy MariaDB

1. Upewnij się, że MariaDB jest zainstalowana i uruchomiona jako usługa (lokalnie).
2. Zaloguj się do klienta MariaDB (np. z poziomu terminala lub narzędzia graficznego).
3. Utwórz bazę danych, np. `library_db`.
4. Utwórz użytkownika aplikacyjnego (np. `app_user`) i nadaj mu uprawnienia do bazy `library_db`.
5. Zanotuj parametry połączenia, które wykorzystasz w aplikacji:
   - host (zwykle `localhost`),
   - port (zwykle `3306`),
   - user, password,
   - database.

Wynik: Baza `library_db` istnieje, a użytkownik aplikacyjny ma uprawnienia do wykonywania operacji na tej bazie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 2

Utworzenie schematu: tabela i klucze

Aby wykonywać CRUD, musisz posiadać tabelę z poprawnie zdefiniowanym kluczem głównym.

1. W bazie `library_db` utwórz tabelę `books`.
2. Zdefiniuj pola:
    - `id` jako klucz główny (autoinkrementacja),
    - `isbn` jako pole unikalne (zapobiega duplikatom),
    - `title` i `author` jako pola tekstowe,
    - opcjonalnie `created_at` jako znacznik czasu.
3. Sprawdź, czy tabela została utworzona i czy posiada poprawne ograniczenia (PRIMARY KEY, UNIQUE).

Wynik: W bazie istnieje tabela `books`, a pole `id` jest kluczem głównym, zaś `isbn` ma ograniczenie unikalności.

Ćwiczenie 3

Przygotowanie aplikacji serwerowej i połączenia z bazą

W tym kroku skonfigurujesz aplikację serwerową, która będzie łączyć się z MariaDB. Założenie: aplikacja działa lokalnie i udostępnia API na wybranym porcie.

1. Utwórz katalog projektu (np. `crud-mariadb`) oraz plik startowy serwera.
2. Zainicjalizuj projekt (np. menedżerem pakietów) i zainstaluj:
    - bibliotekę do tworzenia API (np. framework serwerowy),
    - sterownik do połączenia z MariaDB (kompatybilny z MySQL).
3. Utwórz moduł/plik konfiguracji połączenia (np. `db.js`), który:
    - tworzy połączenie lub pulę połączeń,
    - zawiera parametry host/user/password/database.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

4. Dodaj test połączenia (np. wykonanie prostego zapytania `SELECT 1`).

Wynik: Po uruchomieniu serwera aplikacja nawiązuje połączenie z bazą (brak błędów), a testowe zapytanie zwraca wynik.

Ćwiczenie 4

CREATE – dodawanie rekordu do bazy (INSERT)

1. Zdefiniuj endpoint POST np. `POST /api/books`.
2. Zaimplementuj walidację danych wejściowych (minimum):
   - `isbn` niepuste,
   - `title` min. 3 znaki,
   - `author` min. 3 znaki.
3. Zbuduj zapytanie SQL `INSERT INTO books (isbn, title, author) VALUES (?, ?, ?)` z użyciem parametrów (zapytania parametryzowane).
4. Wykonaj zapytanie w bazie i odczytaj identyfikator nowego rekordu.
5. Zwróć odpowiedź JSON (np. nowy rekord lub `id` + komunikat) oraz kod statusu (np. 201).
6. Obsłuż błędy:
   - duplikat `isbn` → błąd walidacji/konflikt,
   - brak danych → 400.

Wynik: Po wysłaniu `POST /api/books` rekord jest zapisany w tabeli `books`, a klient otrzymuje potwierdzenie (np. `id` nowej książki).

Ćwiczenie 5

READ – odczyt listy rekordów (SELECT)

1. Zdefiniuj endpoint GET np. `GET /api/books`.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

2. Wykonaj zapytanie SQL `SELECT id, isbn, title, author FROM books ORDER BY id DESC`.
3. Zwróć wyniki jako tablicę JSON.
4. Dodaj obsługę sytuacji, gdy tabela jest pusta (zwróć pustą tablicę, nie błąd).

Wynik: Wywołanie `GET /api/books` zwraca listę rekordów w formacie JSON (lub pustą listę, jeśli brak danych).

Ćwiczenie 6

READ – odczyt pojedynczego rekordu po identyfikatorze

1. Zdefiniuj endpoint GET np. `GET /api/books/:id`.
2. Sprawdź poprawność parametru `id` (czy jest liczbą dodatnią).
3. Wykonaj zapytanie SQL `SELECT id, isbn, title, author FROM books WHERE id = ?`.
4. Jeśli rekord nie istnieje – zwróć 404 z komunikatem.

Wynik: Dla istniejącego `id` endpoint zwraca rekord; dla nieistniejącego – 404 z informacją o braku zasobu.

Ćwiczenie 7

UPDATE – edycja rekordu (UPDATE)

Aktualizacja powinna zmieniać wybrane pola rekordu i informować klienta o wyniku.

1. Zdefiniuj endpoint PUT np. `PUT /api/books/:id`.
2. Zweryfikuj `id` oraz dane wejściowe (np. tytuł/autor niepuste).
3. Sprawdź, czy rekord istnieje (np. przez SELECT) – dzięki temu możesz zwrócić 404 przed aktualizacją.
4. Wykonaj zapytanie SQL `UPDATE books SET isbn = ?, title = ?, author = ? WHERE id = ?`.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

5. Zwróć odpowiedź (np. zaktualizowany rekord lub komunikat) i kod 200.

Wynik: Po `PUT /api/books/:id` rekord w bazie ma zaktualizowane wartości, co można potwierdzić ponownym odczytem.

Ćwiczenie 8

DELETE – usuwanie rekordu (DELETE)

Celem jest usunięcie wskazanej książki z bazy.

1. Zdefiniuj endpoint DELETE np. `DELETE /api/books/:id`.
2. Zweryfikuj `id`.
3. Wykonaj zapytanie SQL `DELETE FROM books WHERE id = ?`.
4. Jeśli liczba zmienionych wierszy wynosi 0 – zwróć 404 (rekord nie istniał).
5. W przypadku sukcesu zwróć 200/204.

Wynik: Po wywołaniu `DELETE /api/books/:id` rekord znika z tabeli `books`, a endpoint zwraca informację o powodzeniu.

Ćwiczenie 9

Testowanie CRUD i weryfikacja w bazie

Aby potwierdzić poprawność implementacji, testuj API w sposób powtarzalny i kontrolowany.

1. Wykonaj sekwencję testów (w narzędziu do API lub wierszu poleceń):
   - dodaj 2–3 rekordy (POST),
   - odczytaj listę (GET),
   - odczytaj rekord po id (GET),
   - zaktualizuj rekord (PUT),
   - usuń rekord (DELETE).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

2. Po każdej operacji wykonaj w MariaDB zapytania kontrolne:
   - `SELECT * FROM books`; (stan tabeli),
   - sprawdź, czy `isbn` nie powtarza się.
3. Zwróć uwagę na statusy HTTP i treść odpowiedzi JSON – powinny być spójne.

Wynik: Wyniki API są zgodne z tym, co faktycznie znajduje się w tabeli `books`, a operacje CRUD można jednoznacznie potwierdzić zapytaniami SQL.

Ćwiczenia samodzielne

Ćwiczenie 1

Dodaj endpoint `GET /api/books?author=...` filtrujący książki po autorze.

Ćwiczenie 2

Dodaj walidację: ISBN musi mieć określony format (np. 10/13 znaków), a przy błędzie zwracaj 400 z czytelnym komunikatem.

Ćwiczenie 3

Dodaj paginację listy: `GET /api/books?page=1&limit=10`.

Ćwiczenie 4

Dodaj pole `year` i umożliw edycję tylko wybranych pól (np. PATCH).

Ćwiczenie 5

Dodaj drugą tabelę `loans` (wypożyczenia) i relację do `books` (klucz obcy). Zaimplementuj endpoint zwracający książki wraz z informacją, czy są wypożyczone.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 6

Zaimplementuj transakcję: dodanie książki oraz wpis do tabeli logów (np. `book_logs`) w jednej operacji – jeśli jedna część się nie uda, całość ma zostać wycofana.

Pytania pomocnicze

1. Na czym polega model CRUD i jak mapuje się go na operacje SQL?
2. Czym różni się klucz główny od pola unikalnego i dlaczego oba są istotne w projektowaniu tabel?
3. Dlaczego należy stosować zapytania parametryzowane zamiast sklejania SQL jako tekstu?
4. Jaką rolę pełni warstwa serwerowa w walidacji danych i obsłudze błędów, a jaką baza danych?
5. Jak interpretować wynik operacji UPDATE/DELETE, gdy liczba zmienionych wierszy wynosi 0?
6. Jakie kody statusu HTTP są najbardziej typowe dla operacji CRUD i kiedy należy je stosować?
7. Jak sprawdzić, czy dane zwracane przez API są zgodne z tym, co znajduje się w bazie?
8. Jakie ryzyka (bezpieczeństwo, spójność) pojawiają się, gdy aplikacja nie waliduje danych lub nie używa transakcji?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Podsumowanie

W ramach laboratorium skonfigurowano bazę danych MariaDB, utworzono schemat tabeli oraz połączono aplikację serwerową z bazą w celu wykonywania zapytań SQL. Następnie zaimplementowano komplet operacji CRUD jako endpointy API, co pozwoliło zrozumieć, jak dane są tworzone, odczytywane, aktualizowane i usuwane w relacyjnej bazie danych. Kluczowe okazały się walidacja danych wejściowych, stosowanie zapytań parametryzowanych oraz testowanie zmian poprzez porównanie odpowiedzi API ze stanem tabel w MariaDB. Zdobyte umiejętności przekładają się bezpośrednio na praktyczne projektowanie aplikacji webowych, w których warstwa serwerowa komunikuje się z relacyjną bazą danych i udostępnia bezpieczne, przewidywalne API.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 9

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Uwierzytelnianie użytkowników stanowi jeden z kluczowych elementów współczesnych aplikacji internetowych. Jego głównym celem jest potwierdzenie tożsamości użytkownika oraz zapewnienie, że dostęp do określonych zasobów systemu mają wyłącznie osoby uprawnione. W praktyce proces ten realizowany jest najczęściej poprzez mechanizmy rejestracji, logowania oraz zarządzania sesją, które razem tworzą spójny system kontroli dostępu.

Rejestracja użytkownika polega na zebraniu danych identyfikujących (np. login, adres e-mail, hasło) i zapisaniu ich w trwałym repozytorium po stronie serwera. Szczególnie istotnym aspektem tego procesu jest bezpieczne przetwarzanie haseł – w poprawnie zaprojektowanym systemie hasła nigdy nie są przechowywane w postaci jawnej, lecz w formie skrótów kryptograficznych. Logowanie użytkownika to z kolei proces weryfikacji wprowadzonych danych na podstawie informacji zapisanych wcześniej w systemie.

Po pomyślnym uwierzytelnieniu aplikacja musi „pamiętać" zalogowanego użytkownika pomiędzy kolejnymi żądaniami HTTP, które same w sobie są bezstanowe. Do tego celu wykorzystuje się mechanizm sesji, oparty najczęściej na identyfikatorze sesji przechowywanym w pliku cookie po stronie przeglądarki. Każde kolejne żądanie użytkownika zawiera ten identyfikator, co pozwala serwerowi powiązać je z odpowiednim kontekstem sesji.

W niniejszym laboratorium student krok po kroku zaprojektuje i zaimplementuje prosty system uwierzytelniania użytkowników. Wykorzystane zostaną formularze HTML, obsługa zdarzeń i walidacja danych po stronie klienta, komunikacja asynchroniczna z serwerem oraz mechanizmy sesji. Ćwiczenie pozwala zrozumieć zarówno techniczne aspekty logowania, jak i podstawowe zasady bezpieczeństwa aplikacji webowych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest zaprojektowanie i implementacja prostego systemu uwierzytelniania użytkowników w aplikacji webowej. Student nauczy się obsługi formularzy rejestracji i logowania, komunikacji z serwerem oraz zarządzania sesją użytkownika. Efektem końcowym będzie działający mechanizm logowania i wylogowania.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie interfejsu użytkownika – formularz rejestracji

1. Dodaj element `<form>` z unikalnym `id`, np. `registerForm`.
2. Dodaj pola wejściowe wraz z etykietami `<label>`:
   - login (`id="regUsername"`, `name="username"`),
   - e-mail (`id="regEmail"`, `name="email"`, `type="email"`),
   - hasło (`id="regPassword"`, `name="password"`, `type="password"`),
   - potwierdzenie hasła (`id="regPassword2"`, `name="password2"`, `type="password"`).
3. Dodaj przycisk wysyłki `type="submit"` (np. „Zarejestruj”).
4. Pod formularzem dodaj kontener na komunikaty (np. `<div id="registerMessage"></div>`).
5. Ustal konwencję: komunikat sukcesu vs błąd (np. klasy CSS `success`/`error`).
6. Upewnij się, że `label for="..."` wskazuje na poprawne `id`.
7. Ustal kolejność tabulacji (zwykle wynika z kolejności w HTML).

Wynik: Na stronie widoczny jest formularz rejestracji z czterema polami oraz miejscem na komunikaty; po kliknięciu „Zarejestruj” przeglądarka nie zgłasza błędów struktury.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 2

Walidacja danych formularza rejestracji po stronie klienta

1. W JavaScript poczekaj na załadowanie DOM (np. `DOMContentLoaded`).
2. Pobierz formularz przez `getElementById('registerForm')`.
3. Dodaj nasłuchiwanie na zdarzenie `submit`.
4. W funkcji obsługi wykonaj `event.preventDefault()`.
5. Odczytaj wartości z pól (`value`).
6. Dla loginu i e-maila zastosuj `trim()` (usuwa spacje z początku/końca).
7. Sprawdź, czy pola nie są puste.
8. Sprawdź minimalną długość loginu (np. 3 znaki).
9. Sprawdź minimalną długość hasła (np. 8 znaków).
10. Sprawdź zgodność hasła i potwierdzenia.
11. Sprawdź prostą regułę „siły" hasła (np. cyfra + litera).
12. Jeśli pojawi się błąd, wyświetl komunikat w `registerMessage` i przerwij dalsze kroki (np. `return`).
13. Dodatkowo możesz:
    - dodać klasę błędu do konkretnego pola,
    - ustawić fokus na pierwszym błędnym polu.
14. Jeśli wszystkie reguły są spełnione, przejdź do Ćwiczenia 3.

Wynik: Formularz nie wysyła danych, gdy występują błędy (puste pola, zbyt krótkie hasło, niezgodne hasła). Użytkownik widzi jasny komunikat, a poprawne dane przechodzą do etapu wysyłki.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 3

Wysłanie danych rejestracyjnych do serwera

1. Zbuduj obiekt danych, np. `{ username, email, password }`.
2. Nie wysyłaj pola `password2` – służy tylko do walidacji klienta.
3. Wywołaj zapytanie do API (np. `fetch('/api/register', { ... })`).
4. Ustaw nagłówki:
   - `Content-Type: application/json`.
5. Ustaw body jako JSON.
6. Jeśli serwer zwraca sukces:
   - pokaż komunikat „Konto utworzone”,
   - opcjonalnie: przekieruj do logowania.
7. Jeśli serwer zwraca błąd walidacji (np. użytkownik istnieje):
   - wyświetl komunikat zwrócony przez serwer.
8. Zarejestruj nowego użytkownika – powinno się udać.
9. Spróbuj zarejestrować drugi raz ten sam login/e-mail – powinien pojawić się błąd.
10. Spróbuj wysłać puste pola – klient powinien zatrzymać wysyłkę.

Wynik: Po poprawnej rejestracji serwer potwierdza utworzenie użytkownika, a UI pokazuje sukces. W przypadku konfliktu (np. login zajęty) UI pokazuje błąd bez przeładowania strony.

Ćwiczenie 4

Implementacja formularza logowania

1. Dodaj `<form id="loginForm">`.
2. Dodaj pola:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- login lub e-mail (`id="loginId"`, `name="login"`),
- hasło (`id="loginPassword"`, `name="password"`, `type="password"`).

3. Dodaj przycisk `type="submit"` („Zaloguj”).
4. Dodaj kontener komunikatów (np. `<div id="loginMessage"></div>`).
5. Zarejestruj `submit` i wykonaj `preventDefault()`.
6. Pobierz wartości pól.
7. Walidacja minimalna:
   - brak pustych pól,
   - ewentualnie minimalna długość hasła.
8. Wyślij `POST /api/login` z danymi `{ login, password }` w JSON.
9. Obsłuż odpowiedź:
   - sukces: komunikat + przejście do obszaru chronionego,
   - błąd: komunikat „Niepoprawne dane logowania”.

Wynik: Użytkownik może wprowadzić dane logowania, a aplikacja wysyła żądanie do serwera i wyświetla wynik (sukces/błąd) bez przeładowania strony.

Ćwiczenie 5

Weryfikacja danych logowania po stronie serwera

1. Odczytaj `login` i `password` z body.
2. Sprawdź, czy oba pola istnieją – jeśli nie, zwróć błąd walidacji.
3. Wyszukaj użytkownika po loginie lub e-mailu (w zależności od założeń).
4. Jeśli nie znaleziono użytkownika – zwróć błąd (bez ujawniania, czy login istnieje).
5. Porównaj hasło podane przez użytkownika z hasłem zapisanym w systemie.
6. Jeśli hasło jest niepoprawne – zwróć błąd.
7. Wygeneruj identyfikator sesji (losowy, trudny do zgadnięcia).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

8. Zapisz sesję po stronie serwera (np. w pamięci lub w bazie), mapując `sessionId` → `userId`.
9. Ustaw cookie z identyfikatorem sesji w odpowiedzi.
10. Zwróć JSON (np. `{ message: 'Zalogowano' }` oraz ewentualnie bezpieczne dane użytkownika).
11. Ustaw odpowiedni kod statusu.

Wynik: Serwer rozpoznaje poprawne dane logowania, tworzy sesję oraz ustawia cookie z identyfikatorem sesji; klient otrzymuje potwierdzenie logowania.

Ćwiczenie 6

Zarządzanie sesją użytkownika

1. Utwórz przykładowy endpoint wymagający zalogowania, np. `GET /api/me` lub `GET /api/profile`.
2. W odpowiedzi endpoint zwraca dane tylko dla zalogowanego użytkownika.
3. Przy każdym żądaniu do zasobu chronionego odczytaj cookie `sessionId`.
4. Jeśli cookie nie istnieje – zwróć 401 (brak uwierzytelnienia).
5. Sprawdź, czy `sessionId` istnieje w magazynie sesji.
6. Jeśli sesja nie istnieje lub wygasła – zwróć 401.
7. Jeśli istnieje – pobierz `userId` i ustaw kontekst użytkownika (np. w obiekcie request).
8. Na podstawie `userId` pobierz dane użytkownika (np. login, e-mail, rola).
9. Zwróć je w JSON.
10. Wywołaj endpoint chroniony bez logowania – powinien zwrócić 401.
11. Zaloguj się i ponów wywołanie – powinien zwrócić dane użytkownika.
12. Usuń cookie w przeglądarce i ponów wywołanie – powinien wrócić 401.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Wynik: Serwer blokuje dostęp do zasobów chronionych bez sesji (401) i poprawnie udostępnia je po zalogowaniu, rozpoznając użytkownika na podstawie cookie z identyfikatorem sesji.

Ćwiczenie 7

Wylogowanie użytkownika (POST/GET /api/logout)

1. Utwórz endpoint, np. `POST /api/logout`.
2. Endpoint powinien działać niezależnie od UI (możesz go wywołać przyciskiem „Wyloguj").
3. Odczytaj `sessionId` z cookie.
4. Usuń wpis sesji z magazynu sesji.
5. W odpowiedzi ustaw cookie `sessionId` tak, aby zostało usunięte (wyczyszczenie wartości / wygaszenie).
6. Po sukcesie wylogowania ukryj elementy „dla zalogowanych".
7. Pokaż ponownie formularz logowania lub przekieruj na stronę startową.
8. Zaloguj się i wywołaj endpoint chroniony – powinien działać.
9. Wyloguj się.
10. Ponownie wywołaj endpoint chroniony – powinien zwracać 401.

Wynik: Po wylogowaniu sesja przestaje istnieć po stronie serwera, cookie zostaje usunięte, a dostęp do zasobów chronionych jest zablokowany.

Ćwiczenia samodzielne

Ćwiczenie 1

Dodaj walidację adresu e-mail po stronie klienta.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 2

Zaimplementuj blokadę konta po kilku nieudanych próbach logowania.

Ćwiczenie 3

Dodaj obsługę komunikatów błędów zwracanych przez serwer.

Ćwiczenie 4

Rozszerz system o role użytkowników (np. użytkownik / administrator).

Ćwiczenie 5

Zaimplementuj automatyczne wylogowanie po określonym czasie bezczynności.

Pytania pomocnicze

1. Na czym polega proces uwierzytelniania, a czym jest autoryzacja?
2. Dlaczego hasła nie powinny być przechowywane w postaci jawnej?
3. Jaką rolę pełnią sesje w aplikacjach webowych?
4. Jakie są różnice między walidacją po stronie klienta i serwera?
5. Jakie zagrożenia bezpieczeństwa mogą wynikać z błędnej obsługi sesji?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Podsumowanie

W ramach laboratorium zaprojektowano i zaimplementowano podstawowy system uwierzytelniania użytkowników obejmujący rejestrację, logowanie oraz zarządzanie sesjami. Ćwiczenie pozwoliło zrozumieć znaczenie walidacji danych, komunikacji klient–serwer oraz mechanizmów utrzymywania stanu użytkownika. Zdobyta wiedza stanowi fundament do dalszego rozwijania bezpiecznych aplikacji webowych wykorzystujących kontrolę dostępu i ochronę danych użytkowników.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 10

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

W aplikacjach webowych rozróżnia się dwa pojęcia, które często są ze sobą mylone: uwierzytelnianie (authentication) oraz autoryzację (authorization). Uwierzytelnianie odpowiada na pytanie „kim jesteś?" i polega na potwierdzeniu tożsamości użytkownika (np. na podstawie loginu i hasła). Autoryzacja odpowiada natomiast na pytanie „co wolno Ci zrobić?" i polega na sprawdzeniu, czy użytkownik ma uprawnienia do wykonania konkretnej operacji lub uzyskania dostępu do określonego zasobu.

W praktyce brak poprawnie wdrożonej autoryzacji jest jedną z najpoważniejszych podatności. W literaturze bezpieczeństwa webowego często występuje termin broken access control – oznacza on sytuację, w której kontrola dostępu jest błędna, niekompletna lub pomijana, co umożliwia użytkownikom wykonywanie akcji, do których nie powinni mieć uprawnień (np. dostęp do danych innych użytkowników, wykonywanie operacji administracyjnych). Autoryzacja jest więc elementem bezpieczeństwa, który musi działać konsekwentnie na całej ścieżce przetwarzania żądania: od przyjęcia requestu przez serwer, przez sprawdzenie sesji użytkownika, aż po wykonanie operacji na danych.

W tym laboratorium zastosujesz typowe podejście do kontroli dostępu poprzez role oraz uprawnienia. Najczęściej spotykanym modelem jest RBAC (Role-Based Access Control), w którym prawa są przypisane do ról (np. `user`, `admin`), a role przypisuje się użytkownikom. Alternatywnie można spotkać podejście ACL (Access Control Lists), gdzie dla konkretnych zasobów definiuje się listy dozwolonych operacji. Niezależnie od modelu, autoryzacja wymaga jasnych reguł biznesowych, spójnego sposobu ich egzekwowania oraz testów, które potwierdzają, że zasoby chronione są rzeczywiście chronione.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Cel laboratorium

Celem laboratorium jest rozbudowanie aplikacji o mechanizm autoryzacji oparty o role użytkowników oraz egzekwowanie kontroli dostępu na poziomie endpointów API. Student nauczy się odróżniać uwierzytelnianie od autoryzacji, a także implementować reguły dostępu do zasobów oraz operacji. Efektem będzie zestaw endpointów publicznych i chronionych, poprawnie blokowanych dla użytkowników bez uprawnień.

Instrukcja laboratoryjna

W tej części wdrożysz kontrolę dostępu do API, tak aby:

- użytkownik niezalogowany miał dostęp tylko do zasobów publicznych,
- użytkownik zalogowany miał dostęp do zasobów prywatnych (np. „moje dane"),
- użytkownik z rolą admin miał dostęp do zasobów administracyjnych (np. zarządzanie użytkownikami),
- użytkownik miał możliwość modyfikacji tylko „własnych" rekordów (autoryzacja własności).

Wskazane kroki możesz wykonać w oparciu o mechanizm sesji z poprzedniego laboratorium. Jeśli pracujesz tokenami – analogicznie, tylko zamiast cookie z `sessionId` sprawdzasz token w nagłówku.

Ćwiczenie 1

Zdefiniowanie wymagań autoryzacji i ról

1. Sprawdź w kodzie serwera listę tras API (np. `/api/books, /api/books/:id`, `/api/me, /api/login`, `/api/register`).
2. Zapisz je w tabeli w formacie: *metoda + ścieżka + opis funkcji*.
3. Dla każdego endpointu odpowiedz na pytanie: „Czy użytkownik niezalogowany powinien mieć do niego dostęp?"

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

4. Oznacz:
   - publiczne (np. `POST /api/login`, `POST /api/register`, ewentualnie `GET /api/books` jeśli ma być publiczny odczyt),
   - prywatne (wymagają zalogowania, np. `GET /api/me`, `POST /api/books`),
   - admin (wymagają roli admin, np. `GET /api/admin/users`).
5. Ustal minimalny zestaw ról (np. `user`, `admin`).
6. Opisz je jednym zdaniem:
   - `user`: standardowy użytkownik aplikacji,
   - `admin`: użytkownik z prawem zarządzania innymi użytkownikami i/lub zasobami.
7. Wybierz zasób, dla którego wprowadzisz własność (np. `books`).
8. Ustal zasadę: „Użytkownik może edytować/usunąć tylko rekordy, które sam utworzył”.
9. Zanotuj, które operacje będą sprawdzały własność (najczęściej `PUT` i `DELETE`).

Wynik: Gotowa tabela zasad (endpoint → wymagane logowanie/rola/własność). Na jej podstawie będziesz implementowana ochrona oraz testy.

Ćwiczenie 2

Rozszerzenie modelu użytkownika o role (baza danych)

1. Otwórz schemat bazy (np. tabela `users`).
2. Dodaj kolumnę `role` (typ tekstowy) z domyślną wartością `user`.
3. Zaktualizuj istniejących użytkowników, aby mieli ustawioną rolę (jeśli w bazie są już rekordy).
4. W endpointcie rejestracji (np. `POST /api/register`) dopilnuj, aby zapis nowego użytkownika ustawiał `role='user'`.
5. Upewnij się, że klient nie może sam wysłać roli w formularzu rejestracji (rola jest ustalana po stronie serwera).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

6. Utwórz nowego użytkownika i ustaw mu rolę `admin` (wprost w bazie lub specjalnym skryptem).
7. Zanotuj dane logowania admina (do późniejszych testów 401/403/200).

Wynik: W bazie każdy użytkownik posiada `role`, nowe konta mają domyślnie `user`, a w systemie istnieje konto `admin` do weryfikacji autoryzacji.

Ćwiczenie 3

Ustanowienie „kontekstu użytkownika" po uwierzytelnieniu

1. Minimalnie w sesji powinny znaleźć się: `userId` i opcjonalnie `role`.
2. Jeśli trzymasz w sesji tylko `userId`, to rolę doczytujesz z bazy przy każdym żądaniu (lub cache'ujesz w sesji po zalogowaniu).
3. Na początku obsługi requestu odczytaj identyfikator sesji (np. `sessionId` z cookie).
4. Jeśli brak identyfikatora – uznaj użytkownika za niezalogowanego.
5. Jeśli identyfikator jest – sprawdź, czy istnieje w magazynie sesji.
6. Jeżeli sesja jest poprawna:
   - odczytaj `userId` (oraz `role`, jeśli jest w sesji),
   - jeśli `role` nie jest w sesji, pobierz ją z bazy,
   - ustaw `request.user = { id: userId, role }`.
7. Jeżeli sesja jest niepoprawna – nie ustawiaj `request.user`.
8. Utwórz tymczasowy endpoint diagnostyczny (np. `GET /api/whoami`) zwracający `request.user`.
9. Wywołaj go bez logowania (powinien pokazać brak użytkownika).
10. Zaloguj się i wywołaj ponownie (powinien zwrócić id i role).

Wynik: W każdym requestcie serwer potrafi jednoznacznie stwierdzić, czy użytkownik jest zalogowany, i zna jego `id` oraz `role`.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 4

Ochrona zasobów prywatnych: middleware `requireAuth` (401)

1. W middleware sprawdź, czy `request.user` istnieje.
2. Jeśli nie istnieje:
   - zwróć odpowiedź z kodem **401 Unauthorized**,
   - w treści zwróć JSON z krótkim komunikatem (np. `{ error: 'Unauthorized', message: 'Wymagane logowanie' }`),
   - zakończ obsługę żądania.
3. Jeśli `request.user` istnieje – przepuść request do dalszej logiki.
4. Wybierz 2–4 endpointy, które wymagają logowania (np. GET /api/me, `POST /api/books`, `PUT /api/books/:id`, `DELETE /api/books/:id`).
5. Zastosuj middleware przed właściwym handlerem endpointu.
6. Bez logowania wywołaj endpoint prywatny → oczekuj 401.
7. Zaloguj się jako zwykły użytkownik → endpoint powinien działać.

Wynik: Endpointy prywatne są niewidoczne dla niezalogowanych (401), a dostępne dla zalogowanych.

Ćwiczenie 5

Ochrona zasobów administracyjnych: middleware `requireRole` (403) – RBAC

1. Wewnątrz `requireRole` najpierw sprawdź logowanie (wywołaj `requireAuth` lub wykonaj analogiczny check).
2. Następnie porównaj `request.user.role` z wymaganą rolą.
3. Jeśli rola nie pasuje:
   - zwróć 403 Forbidden,
   - odpowiedź JSON (np. `{ error: 'Forbidden', message: 'Brak uprawnień' }`).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

4. Dodaj endpoint np. `GET /api/admin/users`.
5. W handlerze pobierz listę użytkowników z bazy (np. `id`, `username`, `email`, `role`) i zwróć w JSON.
6. Dodaj `requireRole('admin')` do tego endpointu.
7. Bez logowania → 401.
8. Zalogowany user (rola `user`) → 403.
9. Zalogowany admin → 200 i lista użytkowników.

Wynik: Admin widzi zasoby administracyjne (200), zwykły user dostaje 403, a niezalogowany 401.

Ćwiczenie 6

Autoryzacja własności zasobu (ownership) – „użytkownik edytuje tylko swoje dane"

1. W tabeli zasobu (np. `books`) dodaj kolumnę `owner_id`.
2. Ustal typ zgodny z `users.id`.
3. (Opcjonalnie) Dodaj klucz obcy do tabeli `users`.
4. W endpointcie tworzącym rekord (np. `POST /api/books`) dodaj `requireAuth`.
5. Przy zapisie rekordu ustaw `owner_id = request.user.id`.
6. Zwróć w odpowiedzi również informację o właścicielu (opcjonalnie).
7. W endpointcie `PUT /api/books/:id`:
   - dodaj `requireAuth`,
   - pobierz rekord z bazy po `id`.
8. Jeśli rekord nie istnieje → 404.
9. Jeśli istnieje:
   - porównaj `record.owner_id` z `request.user.id`.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

10. Jeśli nie pasuje → 403.
11. Jeśli pasuje → wykonaj aktualizację.
12. Analogicznie w `DELETE /api/books/:id`:
    - pobierz rekord,
    - 404 jeśli brak,
    - 403 jeśli nie właściciel,
    - usuń jeśli właściciel.
13. Zaloguj się jako User A, utwórz rekord.
14. Zaloguj się jako User B i spróbuj edytować/usunąć rekord User A → 403.
15. Zaloguj się ponownie jako User A → edycja/usunięcie działa.

Wynik: Użytkownicy nie mogą modyfikować cudzych rekordów (403), a rekordy mają przypisanego właściciela ustawianego automatycznie po stronie serwera.

Ćwiczenie 7

Zasada „deny by default" i spójne odpowiedzi błędów

1. Wróć do tabeli z Ćwiczenia 1.
2. Sprawdź, czy każdy endpoint prywatny ma `requireAuth`.
3. Sprawdź, czy każdy endpoint admin ma `requireRole('admin')`.
4. Sprawdź, czy endpointy operujące na rekordach mają logikę ownership (jeśli wymagane).
5. Ustal wspólny format JSON dla błędów, np. `{ error, message }`.
6. Upewnij się, że 401 i 403 mają różne znaczenia:
    - 401: „nie jesteś zalogowany",
    - 403: „jesteś zalogowany, ale nie masz praw".

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

7. Nie ujawniaj w komunikatach zbyt wielu szczegółów (np. nie informuj, czy użytkownik istnieje).
8. Loguj szczegóły po stronie serwera (w logach), a użytkownikowi pokazuj komunikat ogólny.

Wynik: Zasoby są domyślnie zablokowane, a API zwraca spójne i bezpieczne odpowiedzi błędów.

Ćwiczenie 8

Testowanie autoryzacji – pełna macierz scenariuszy

1. Przygotuj zestaw kont testowych
   - Konto `User A` (rola `user`).
   - Konto `User B` (rola `user`).
   - Konto `Admin` (rola `admin`).
2. Scenariusze testowe dla endpointów prywatnych (401/200)
   - Bez logowania: wywołaj endpoint prywatny → 401.
   - Zalogowany User A: wywołaj endpoint prywatny → 200.
3. Scenariusze dla endpointów admin (401/403/200)
   - Bez logowania → 401.
   - Zalogowany User A → 403.
   - Zalogowany Admin → 200.
4. Scenariusze własności (403/200)
   - User A tworzy rekord.
   - User B próbuje go edytować/usunąć → 403.
   - User A edytuje/usuwa → 200.
5. Zapisz wyniki do sprawozdania – dla każdego testu zapisz:

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- wywoływany endpoint,
- status HTTP,
- krótki opis, dlaczego taki wynik jest poprawny.

Wynik: Masz kompletny zestaw testów potwierdzający poprawną autoryzację (401/403/200/404).

Ćwiczenia samodzielne

Ćwiczenie 1

Dodaj trzecią rolę editor, która może tworzyć i edytować rekordy, ale nie może usuwać.

Ćwiczenie 2

Oznacz część endpointów jako publiczne „tylko do odczytu" (GET publiczne, POST/PUT/DELETE prywatne).

Ćwiczenie 3

Zaimplementuj endpoint GET /api/admin/logs i zapisuj do bazy logi nieudanych prób dostępu (401/403) wraz z czasem i adresem IP.

Ćwiczenie 4

Dodaj regułę: użytkownik może zmienić swoje hasło tylko po podaniu aktualnego hasła (wymaga zarówno uwierzytelnienia, jak i autoryzacji własności).

Ćwiczenie 5

Zaimplementuj ACL dla wybranego zasobu (np. dokumenty), gdzie właściciel może przyznać innemu użytkownikowi prawo „read" lub „write".

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 6

Przygotuj zestaw testów automatycznych dla autoryzacji (np. sprawdzających 401/403/200) i uruchamiaj je przed oddaniem sprawozdania.

Pytania pomocnicze

1. Czym różni się uwierzytelnianie od autoryzacji i dlaczego oba mechanizmy są potrzebne?
2. Co oznacza podatność „broken access control” i jakie skutki może mieć w aplikacji?
3. Czym różni się model RBAC od ACL i kiedy warto zastosować każdy z nich?
4. Jakie kody statusu HTTP powinny być zwracane przy braku logowania (401) i braku uprawnień (403)?
5. Na czym polega autoryzacja własności zasobu i jak ją poprawnie zaimplementować?
6. Dlaczego zasada „deny by default” zwiększa bezpieczeństwo aplikacji?
7. Jakie ryzyka wiążą się z ujawnianiem zbyt szczegółowych komunikatów błędów?
8. Jakie scenariusze testowe są kluczowe, aby potwierdzić poprawność autoryzacji?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Podsumowanie

W ramach laboratorium rozszerzono aplikację o autoryzację, czyli kontrolę dostępu do zasobów i operacji w zależności od tożsamości oraz roli użytkownika. Zaimplementowano ochronę endpointów prywatnych (wymóg logowania), ochronę zasobów administracyjnych w modelu RBAC oraz autoryzację własności, która blokuje modyfikację danych należących do innych użytkowników. Przeprowadzone testy potwierdziły poprawne działanie kodów 401 i 403 oraz spójne egzekwowanie reguł bezpieczeństwa po stronie serwera. Zdobyte umiejętności są bezpośrednio wykorzystywane w praktycznych projektach, ponieważ większość realnych aplikacji wymaga ograniczania dostępu do danych i operacji w sposób przewidywalny i odporny na próby obejścia.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

POLITECHNIKA LUBELSKA

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 11

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Wprowadzenie, tematyka zajęć

Współczesne aplikacje internetowe opierają się na intensywnej komunikacji pomiędzy przeglądarką użytkownika a serwerem. W klasycznym modelu synchronizacji każda interakcja użytkownika prowadziła do przeładowania całej strony, co było kosztowne i nieefektywne. Rozwiązaniem tego problemu stała się asynchroniczna komunikacja klient–serwer, w której przeglądarka wysyła żądania HTTP w tle, bez przerywania pracy interfejsu użytkownika.

Asynchroniczne programowanie w JavaScript polega na tym, że wywołanie funkcji oraz uzyskanie jej wyniku nie następują w jednym, blokującym kroku. Zamiast tego aplikacja wysyła żądanie (np. do serwera), a odpowiedź jest obsługiwana dopiero wtedy, gdy faktycznie nadejdzie. W praktyce oznacza to, że kod JavaScript musi reagować na zdarzenia zakończenia operacji asynchronicznej, a nie zakładać natychmiastowy wynik działania funkcji. W języku JavaScript stosuje się w tym celu różne mechanizmy, takie jak callbacki, obiekty Promise oraz składnię async/await, które znacząco poprawiają czytelność i kontrolę przepływu programu.

W kontekście komunikacji z serwerem asynchroniczne żądania HTTP umożliwiają pobieranie i wysyłanie danych w formacie JSON, a następnie dynamiczną aktualizację interfejsu użytkownika na podstawie odpowiedzi serwera. Kluczowe znaczenie ma tutaj poprawna obsługa stanów odpowiedzi (sukces, błąd, brak danych), kodów statusu HTTP oraz błędów sieciowych. Równie istotne jest zapewnienie spójności interfejsu, tak aby użytkownik otrzymywał jasną informację o tym, co aktualnie dzieje się w aplikacji.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest nauczenie się realizacji asynchronicznej komunikacji pomiędzy klientem a serwerem z wykorzystaniem JavaScript. Student nauczy się wysyłać żądania HTTP, przetwarzać odpowiedzi serwera oraz reagować na błędy komunikacji. Efektem końcowym będzie interfejs, który dynamicznie aktualizuje się na podstawie danych otrzymywanych z API.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie struktury strony i elementów interfejsu

1. Dodaj przycisk, który zainicjuje pobieranie danych z serwera (np. „Pobierz dane").
2. Dodaj kontener (np. `<div>` lub `<ul>`) przeznaczony na wyświetlanie wyników.
3. Dodaj obszar na komunikaty statusowe (np. „Ładowanie...", „Błąd połączenia").
4. Ustal `id` dla przycisku.
5. Ustal `id` dla kontenera danych.
6. Ustal `id` dla komunikatów.

Wynik: Strona zawiera przycisk inicjujący żądanie, pusty obszar na dane oraz miejsce na komunikaty informacyjne.

Ćwiczenie 2

Wysłanie asynchronicznego żądania GET do serwera

1. W JavaScript pobierz przycisk po `id`.
2. Dodaj nasłuchiwanie na zdarzenie `click`.
3. W funkcji wyświetl komunikat „Ładowanie danych...".
4. Wyślij asynchroniczne żądanie GET do wybranego endpointu API.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

5. Sprawdź, czy odpowiedź zakończyła się sukcesem.
6. Odczytaj dane w formacie JSON.

Wynik: Po kliknięciu przycisku aplikacja wysyła żądanie GET i otrzymuje dane z serwera bez odświeżania strony.

Ćwiczenie 3

Obsługa odpowiedzi serwera i parsowanie danych

1. Zidentyfikuj, czy odpowiedź jest pojedynczym obiektem czy tablicą.
2. Ustal, które pola będą prezentowane w interfejsie.
3. Usuń komunikat „Ładowanie...".
4. Przygotuj dane do wyświetlenia (np. iteracja po tablicy).
5. Jeśli serwer zwróci pustą listę, wyświetl komunikat „Brak danych do wyświetlenia".

Wynik: Dane otrzymane z serwera są poprawnie rozpoznane i przygotowane do prezentacji w interfejsie.

Ćwiczenie 4

Dynamiczna aktualizacja interfejsu użytkownika

1. Usuń stare elementy z obszaru wyników.
2. Dla każdego elementu danych utwórz nowy element DOM.
3. Wypełnij go treścią (np. tekst, nagłówki).
4. Dołącz nowo utworzone elementy do kontenera.

Wynik: Interfejs użytkownika jest aktualizowany dynamicznie na podstawie danych z serwera.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 5

Wysyłanie danych do serwera (żądanie POST)

1. Dodaj pola formularza (np. nazwa, opis).
2. Dodaj przycisk wysyłki.
3. Zablokuj domyślne wysyłanie formularza.
4. Pobierz wartości z pól formularza.
5. Przygotuj obiekt danych.
6. Wyślij go do serwera w formacie JSON.
7. Odbierz odpowiedź serwera.

Wynik: Dane formularza są wysyłane do serwera asynchronicznie, a odpowiedź jest przetwarzana po stronie klienta.

Ćwiczenie 6

Obsługa błędów komunikacji i kodów odpowiedzi

1. Sprawdź kody statusu odpowiedzi.
2. Rozróżnij błędy klienta i serwera.
3. Przechwyć wyjątki związane z brakiem połączenia.
4. Wyświetl komunikat dla użytkownika.
5. Ukryj stan ładowania.
6. Pokaż informację o niepowodzeniu operacji.

Wynik: Aplikacja reaguje poprawnie na błędy serwera i problemy sieciowe, informując użytkownika o sytuacji.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenia samodzielne

Ćwiczenie 1

Dodaj wskaźnik ładowania (spinner) widoczny podczas oczekiwania na odpowiedź serwera.

Ćwiczenie 2

Rozszerz obsługę GET o parametry zapytania (np. filtrowanie wyników).

Ćwiczenie 3

Zaimplementuj żądanie PUT lub DELETE i zaktualizuj interfejs po jego wykonaniu.

Ćwiczenie 4

Dodaj obsługę wielu równoległych żądań i wyświetl ich wyniki po zakończeniu.

Ćwiczenie 5

Zastosuj składnię async/await zamiast klasycznej obsługi Promise.

Pytania pomocnicze

1. Na czym polega asynchroniczna komunikacja klient–serwer?
2. Dlaczego asynchroniczne żądania nie blokują interfejsu użytkownika?
3. Jakie są różnice między żądaniami GET i POST w kontekście komunikacji z API?
4. Jaką rolę pełnią kody statusu HTTP w obsłudze odpowiedzi serwera?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

5. Dlaczego obsługa błędów jest kluczowa w komunikacji asynchronicznej?

Podsumowanie

W ramach laboratorium zrealizowano komunikację pomiędzy klientem a serwerem z wykorzystaniem asynchronicznych żądań HTTP. Student nauczył się wysyłać żądania GET i POST, obsługiwać odpowiedzi serwera oraz dynamicznie aktualizować interfejs użytkownika bez przeładowania strony. Szczególną uwagę poświęcono obsłudze błędów i stanów pośrednich, takich jak ładowanie danych. Zdobyte umiejętności stanowią podstawę do budowy nowoczesnych, responsywnych aplikacji webowych, które efektywnie reagują na dane pochodzące z serwera.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 12

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Wraz ze wzrostem złożoności aplikacji internetowych oraz ilości przetwarzanych danych, zagadnienia związane z wydajnością stają się jednym z kluczowych elementów procesu wytwarzania oprogramowania. Nawet poprawnie działająca funkcjonalnie aplikacja może być nieużyteczna, jeśli czas odpowiedzi serwera jest zbyt długi lub jeśli baza danych nie radzi sobie z obsługą większej liczby zapytań. Jednym z najczęstszych źródeł problemów wydajnościowych są nieoptymalne zapytania do bazy danych.

Optymalizacja zapytań polega na takim formułowaniu instrukcji SQL oraz struktur danych, aby baza danych mogła wykonać je możliwie najszybciej i przy jak najmniejszym zużyciu zasobów. Obejmuje to m.in. dobór odpowiednich typów danych, stosowanie indeksów, ograniczanie liczby przetwarzanych rekordów oraz świadome korzystanie z klauzul takich jak WHERE, JOIN, ORDER BY czy LIMIT. Istotną rolę odgrywa również analiza planów wykonania zapytań, która pozwala zrozumieć, w jaki sposób silnik bazy danych interpretuje i realizuje polecenia SQL.

Z punktu widzenia aplikacji internetowej wydajność bazy danych bezpośrednio wpływa na szybkość odpowiedzi API, płynność interfejsu użytkownika oraz ogólne wrażenia użytkownika końcowego. Dlatego praca nad wydajnością nie jest jednorazowym działaniem, lecz procesem ciągłym, obejmującym monitorowanie, testowanie i stopniowe usprawnianie zarówno warstwy serwerowej, jak i bazy danych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem ćwiczenia jest nauczenie się identyfikowania problemów wydajnościowych związanych z zapytaniami do bazy danych. Student pozna podstawowe techniki optymalizacji zapytań SQL oraz sposoby analizy ich wykonania. Efektem będzie poprawa czasu odpowiedzi aplikacji poprzez świadome modyfikacje zapytań i struktury bazy danych.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie przykładowej bazy danych i danych testowych

1. Utwórz nową bazę danych przeznaczoną do testów wydajności.
2. Utwórz tabelę zawierającą dane (np. użytkownicy, produkty lub zamówienia).
3. Dodaj co najmniej kilkaset lub kilka tysięcy rekordów.
4. Upewnij się, że dane są zróżnicowane (różne wartości pól).

Wynik: Baza danych zawiera tabelę z odpowiednio dużą liczbą rekordów, umożliwiającą obserwację różnic w wydajności zapytań.

Ćwiczenie 2

Wykonanie i analiza podstawowego zapytania SELECT

1. Wykonaj zapytanie pobierające wszystkie rekordy z tabeli.
2. Zanotuj czas wykonania zapytania.
3. Dodaj klauzulę WHERE, aby filtrować dane.
4. Porównaj czas wykonania zapytania z poprzednim wynikiem.

Wynik: Student obserwuje, że ograniczenie liczby przetwarzanych rekordów wpływa na skrócenie czasu wykonania zapytania.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 3

Analiza planu wykonania zapytania

1. Skorzystaj z polecenia analizy zapytania (np. EXPLAIN).
2. Odczytaj informacje o sposobie przetwarzania danych.
3. Sprawdź, czy zapytanie wykonuje pełne skanowanie tabeli.
4. Zwróć uwagę na brak indeksów.

Wynik: Student potrafi wskazać elementy zapytania, które mogą powodować obniżenie wydajności.

Ćwiczenie 4

Zastosowanie indeksów

1. Wybierz kolumnę często używaną w klauzuli WHERE.
2. Utwórz indeks dla tej kolumny.
3. Uruchom to samo zapytanie co wcześniej.
4. Porównaj plan wykonania i czas odpowiedzi.

Wynik: Zapytanie wykonuje się szybciej, a plan wykonania wskazuje użycie indeksu.

Ćwiczenie 5

Optymalizacja zapytań z użyciem JOIN

1. Wykonaj zapytanie z użyciem JOIN.
2. Zanotuj czas wykonania.
3. Dodaj indeksy na kolumnach wykorzystywanych w relacji.
4. Ponownie wykonaj zapytanie.

Wynik: Student zauważa wpływ indeksów na wydajność zapytań łączących wiele tabel.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 6

Ocena wpływu zapytań na aplikację

1. Zaimplementuj endpoint, który wykonuje analizowane zapytanie.
2. Zmierz czas odpowiedzi API.
3. Zestaw czasy odpowiedzi.
4. Wyciągnij wnioski dotyczące wpływu bazy danych na wydajność aplikacji.

Wynik: Student rozumie, jak optymalizacja zapytań SQL przekłada się na szybkość działania aplikacji internetowej.

Ćwiczenia samodzielne

Ćwiczenie 1

Porównaj wydajność zapytania z i bez klauzuli LIMIT.

Ćwiczenie 2

Sprawdź wpływ sortowania (ORDER BY) na czas wykonania zapytania.

Ćwiczenie 3

Zoptymalizuj zapytanie zawierające wiele warunków w klauzuli WHERE.

Ćwiczenie 4

Zaprojektuj dodatkowy indeks i oceń jego wpływ na różne zapytania.

Ćwiczenie 5

Przeanalizuj, kiedy nadmiar indeksów może pogorszyć wydajność.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Pytania pomocnicze

1. Co rozumiemy przez wydajność aplikacji internetowej?
2. Dlaczego nieoptymalne zapytania SQL są częstą przyczyną problemów wydajnościowych?
3. Jaką rolę pełnią indeksy w bazie danych?
4. W jaki sposób plan wykonania zapytania pomaga w jego optymalizacji?
5. Jak optymalizacja bazy danych wpływa na działanie API?

Podsumowanie

Podczas laboratorium przeanalizowano wpływ zapytań SQL na wydajność aplikacji internetowej oraz poznano podstawowe techniki ich optymalizacji. Student nauczył się mierzyć czas wykonania zapytań, analizować plany ich realizacji oraz stosować indeksy w celu poprawy wydajności. Ćwiczenie pozwoliło zrozumieć zależność pomiędzy bazą danych a szybkością odpowiedzi aplikacji serwerowej. Zdobyta wiedza może być bezpośrednio wykorzystana w praktyce przy projektowaniu i rozwijaniu wydajnych systemów informatycznych.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 13

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Wraz ze wzrostem złożoności aplikacji internetowych rośnie ryzyko wprowadzania błędów, które nie zawsze są widoczne na pierwszy rzut oka. Ręczne testowanie aplikacji, polegające na sprawdzaniu jej działania w przeglądarce lub poprzez wywoływanie endpointów API, jest czasochłonne i podatne na pominięcie istotnych scenariuszy. Z tego względu w profesjonalnym procesie wytwarzania oprogramowania kluczową rolę odgrywają testy automatyczne, które pozwalają w sposób powtarzalny i jednoznaczny weryfikować poprawność działania aplikacji.

Jednym z podstawowych rodzajów testów są testy jednostkowe, których celem jest sprawdzanie pojedynczych fragmentów kodu, takich jak funkcje, moduły lub klasy. W przypadku aplikacji webowych szczególne znaczenie mają testy jednostkowe warstwy serwerowej, ponieważ odpowiada ona za logikę biznesową, walidację danych oraz komunikację z bazą danych. Testy te umożliwiają szybkie wykrywanie błędów w logice aplikacji jeszcze przed jej uruchomieniem w środowisku produkcyjnym.

Oprócz testów warstwy serwerowej istotne jest również testowanie interfejsu użytkownika, które pozwala sprawdzić, czy elementy strony reagują poprawnie na działania użytkownika oraz czy prezentowane dane są zgodne z oczekiwaniami. W tym przypadku testy często wykonywane są w środowisku przeglądarki i obejmują zarówno kod JavaScript, jak i interakcje z elementami DOM.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Cel laboratorium

Celem laboratorium jest nauczenie się tworzenia i uruchamiania testów jednostkowych dla aplikacji internetowej. Student pozna podstawy testowania logiki serwerowej oraz testowania interfejsu użytkownika. Efektem ćwiczenia będzie umiejętność weryfikowania poprawności działania aplikacji za pomocą testów automatycznych.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie projektu do testowania (backend)

1. W katalogu projektu utwórz folder tests/ (lub test/).
2. Wewnątrz przygotuj dwa podfoldery:
   - `tests/server/` – testy backendu,
   - `tests/ui/` – testy interfejsu.
3. Zainstaluj bibliotekę uruchamiającą testy (runner) – np. Mocha.
4. Zainstaluj bibliotekę asercji – np. Chai.
5. Zainstaluj narzędzie do testowania HTTP – np. Supertest (ułatwia testowanie endpointów).
6. W pliku konfiguracyjnym projektu (np. w sekcji `scripts`) dodaj polecenia:
   - `test:server` – uruchamia testy serwera,
   - `test:ui` – uruchamia testy UI,
   - `test` – uruchamia wszystko.
7. Dodaj zmienną środowiskową `NODE_ENV=test` (lub analogiczną), aby serwer wiedział, że działa w trybie testowym.
8. W trybie testowym:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

- nie uruchamiaj serwera „na stałe” (nie nasłuchuj portu),
- eksportuj aplikację (np. obiekt `app`), aby testy mogły ją importować.

Przykład (Express): eksport aplikacji bez `listen()`

```
// server/app.js
import express from "express";

export const app = express();
app.use(express.json());

app.get("/api/health", (req, res) => {
     res.json({ status: "ok" });
});

// server/index.js
import { app } from "./app.js";

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server on ${PORT}`));
```

Wynik: Projekt ma uporządkowaną strukturę testów, a aplikacja serwerowa może być importowana w testach bez uruchamiania stałego nasłuchu na porcie.

Ćwiczenie 2

Wybór „jednostki” do testów i zaplanowanie przypadków testowych

1. Wybierz fragment kodu, który ma jasne wejście i wyjście.
2. Dobry przykład: walidacja danych wejściowych dla endpointu `POST /api/books`.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Przykład funkcji walidującej

```
// server/validators/bookValidator.js

export function validateBook(input) {

     const errors = [];

     if   (!input?.isbn   ||   String(input.isbn).trim()   ===   "")
errors.push("isbn_required");

     if  (!input?.title  ||  String(input.title).trim().length  <  3)
errors.push("title_min_3");

     if  (!input?.author  ||  String(input.author).trim().length  <  3)
errors.push("author_min_3");

     return { ok: errors.length === 0, errors };


}
```

3. Zapisz przypadki testowe (minimum)
   - Przypadek poprawny: komplet danych.
   - Przypadek błędny: brak ISBN.
   - Przypadek błędny: za krótki tytuł.
   - Przypadek błędny: za krótki autor.

Wynik: Masz wybraną jednostkę testową oraz listę scenariuszy, które będą pokryte testami.

Ćwiczenie 3

Implementacja testów jednostkowych dla warstwy serwerowej (funkcje)

1. Utwórz `tests/server/bookValidator.test.js`.
2. Zaimportuj `validateBook`.
3. Wywołaj `validateBook` poprawnymi danymi.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

4. Sprawdź, że `ok === true` i lista błędów jest pusta.
5. Dla każdego scenariusza błędnego:
   - wywołaj `validateBook` wadliwymi danymi,
   - sprawdź, że `ok === false`,
   - sprawdź, że `errors` zawiera konkretny kod błędu.

Przykładowe testy (Mocha + Chai)

```
// tests/server/bookValidator.test.js
import { expect } from "chai";
import { validateBook } from
     "../../server/validators/bookValidator.js";
describe("validateBook", () => {
     it("should accept valid book", () => {
          const result = validateBook({ isbn: "123", title: "Clean
Code", author: "Martin" });
          expect(result.ok).to.equal(true);
          expect(result.errors).to.deep.equal([]);
     });

     it("should reject missing isbn", () => {
          const result = validateBook({ title: "Clean Code", author:
"Martin" });
          expect(result.ok).to.equal(false);
          expect(result.errors).to.include("isbn_required");
     });

     it("should reject too short title", () => {
```


Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
          const result = validateBook({ isbn: "123", title: "aa", author: "Martin" });
          expect(result.ok).to.equal(false);
          expect(result.errors).to.include("title_min_3");
     });
});
```

6. Uruchom test:server.
7. Jeśli testy nie przechodzą – popraw kod lub testy.

Wynik: Testy jednostkowe przechodzą i jednoznacznie potwierdzają poprawność walidacji danych wejściowych.

Ćwiczenie 4

Testowanie endpointów API (warstwa serwerowa)

1. Przygotuj endpoint do testów (przykład: `POST /api/books`)
   - W handlerze endpointu:
     - wykonaj walidację `validateBook`,
     - przy błędzie zwróć 400 z JSON,
     - przy sukcesie zwróć 201 z utworzonym obiektem.

Przykładowy endpoint

```
// server/app.js (fragment)
import { validateBook } from "./validators/bookValidator.js";

app.post("/api/books", (req, res) => {
     const { ok, errors } = validateBook(req.body);
     if (!ok) return res.status(400).json({ error: "validation", errors });
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    // w prawdziwej aplikacji zapis do bazy; tu: symulacja
    const created = { id: 1, ...req.body };
    return res.status(201).json(created);
});
```

2. Utwórz plik `tests/server/booksApi.test.js`.
3. Zaimportuj `app`.
4. Wyślij żądanie POST z poprawnymi danymi i sprawdź:
   - status 201,
   - obecność oczekiwanych pól w odpowiedzi.
5. Wyślij żądanie POST z błędnymi danymi i sprawdź:
   - status 400,
   - poprawny format błędu.

Przykładowe testy API

```
// tests/server/booksApi.test.js
import request from "supertest";
import { expect } from "chai";
import { app } from "../../server/app.js";

describe("Books API", () => {
    it("POST /api/books should create book", async () => {
        const res = await request(app)
            .post("/api/books")
            .send({ isbn: "123", title: "Clean Code", author: "Martin" })
            .set("Content-Type", "application/json");
```

```
            expect(res.status).to.equal(201);
            expect(res.body).to.have.property("id");
            expect(res.body.title).to.equal("Clean Code");
      });

      it("POST /api/books should reject invalid input", async () => {
            const res = await request(app)
                  .post("/api/books")
                  .send({ isbn: "", title: "aa", author: "x" })
                  .set("Content-Type", "application/json");

            expect(res.status).to.equal(400);
            expect(res.body).to.have.property("error", "validation");
            expect(res.body.errors).to.include("isbn_required");
      });
});
```

6. Uruchom test:server.
7. Jeśli test nie przechodzi:
   - sprawdź statusy HTTP,
   - sprawdź strukturę JSON,
   - sprawdź, czy app poprawnie eksportujesz.

Wynik: Testy endpointów potwierdzają poprawne zachowanie API (statusy, format odpowiedzi, walidacja).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 5

Przygotowanie środowiska do testów interfejsu użytkownika (UI)

1. Wybierz jeden moduł UI, np. formularz dodawania książki i listę wyników.
2. Ustal zachowania, które testujesz.
   - Czy formularz blokuje wysyłkę, gdy pola są puste?
   - Czy po dodaniu elementu lista aktualizuje się w DOM?
   - Czy komunikaty błędów są wyświetlane?
3. Przygotuj testowy plik HTML dla UI – utwórz plik np. `tests/ui/index.html`.
4. Dołącz:
   - bibliotekę testową UI (np. QUnit),
   - plik z kodem UI (Twoje skrypty),
   - plik z testami.

Wynik: Istnieje oddzielny plik uruchamiający testy UI w przeglądarce.

Ćwiczenie 6

Implementacja testów UI (DOM + zdarzenia) – przykład QUnit

1. W testach wstaw do dokumentu minimalną strukturę:
   - formularz z polami,
   - kontener na listę,
   - kontener na komunikaty.
2. Ważne: testy nie powinny zależeć od „pełnej strony” – tylko od minimalnych elementów.
3. Zasymuluj kliknięcie wysłania formularza z pustymi polami.
4. Sprawdź, czy pojawił się komunikat błędu.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

5. Wywołaj funkcję UI odpowiedzialną za renderowanie listy (np. `renderBooks(books)`),
6. Sprawdź, czy w kontenerze pojawiła się oczekiwana liczba elementów.

Przykładowe funkcje UI (minimalne)

```
// ui/booksView.js
export function renderBooks(listEl, books) {
	listEl.innerHTML = "";
	for (const b of books) {
		const li = document.createElement("li");
		li.textContent = `${b.title} - ${b.author}`;
		listEl.appendChild(li);
	}
}
```

Przykładowe testy QUnit

```
// tests/ui/booksView.test.js
QUnit.module("Books UI", (hooks) => {
	let listEl;

	hooks.beforeEach(() => {
		document.getElementById("qunit-fixture").innerHTML = '<ul id="booksList"></ul>';
		listEl = document.getElementById("booksList");
	});

	QUnit.test("renderBooks renders list items", (assert) => {
		renderBooks(listEl, [
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
                { title: "Clean Code", author: "Martin" },
                { title: "Refactoring", author: "Fowler" },
            ]);

            assert.equal(listEl.querySelectorAll("li").length, 2);
            assert.ok(listEl.textContent.includes("Clean Code"));
        });
});
```

7. Jeśli UI wysyła żądania do serwera, w testach **zamockuj** tę funkcję:
   - zamiast prawdziwego `fetch` zwróć przygotowaną odpowiedź.
8. Sprawdź, czy po „udanej odpowiedzi" UI zaktualizowało DOM.

Wynik: Testy UI potwierdzają, że elementy DOM są poprawnie tworzone/aktualizowane i że UI reaguje na zdarzenia zgodnie z wymaganiami.

Ćwiczenie 7

Analiza wyników testów i iteracyjne poprawki (regresja)

1. Uruchom testy backendu.
2. Uruchom testy UI.
3. Celowo zepsuj jedną regułę walidacji (np. usuń sprawdzanie `isbn_required`).
4. Uruchom testy ponownie.
5. Odczytaj, który test nie przeszedł.
6. Sprawdź, jak test wskazuje miejsce problemu (nazwa testu + asercja).
7. Przywróć poprawną walidację.
8. Uruchom testy ponownie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Wynik: Student widzi, że testy wykrywają regresje, przyspieszają diagnozę problemów i potwierdzają poprawność naprawy.

Ćwiczenia samodzielne

Ćwiczenie 1

Dodaj testy jednostkowe dla innej funkcji warstwy serwerowej.

Ćwiczenie 2

Rozszerz istniejące testy o dodatkowe przypadki błędne.

Ćwiczenie 3

Zaprojektuj test interfejsu użytkownika dla nowego elementu UI.

Ćwiczenie 4

Przeanalizuj, które fragmenty aplikacji są trudne do przetestowania i dlaczego.

Ćwiczenie 5

Spróbuj zaprojektować test w podejściu test-driven development.

Pytania pomocnicze

1. Czym różnią się testy jednostkowe od testów manualnych?
2. Dlaczego testy warstwy serwerowej są szczególnie istotne?
3. Jaką rolę pełnią testy sytuacji błędnych?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

4. W jaki sposób testy automatyczne wspierają rozwój aplikacji?

5. Jakie są ograniczenia testów interfejsu użytkownika?

Podsumowanie

W trakcie laboratorium student zapoznał się z podstawami testowania aplikacji internetowych oraz z rolą testów jednostkowych w procesie wytwarzania oprogramowania. Zrealizowane zadania pozwoliły na praktyczne sprawdzenie logiki serwerowej oraz zachowania interfejsu użytkownika. Ćwiczenie uświadamia znaczenie testów automatycznych jako narzędzia wspierającego jakość, stabilność i dalszy rozwój aplikacji.

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

POLITECHNIKA LUBELSKA

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 14

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Integracja aplikacji internetowych z zewnętrznymi usługami stanowi jeden z kluczowych elementów współczesnego tworzenia oprogramowania. Większość nowoczesnych systemów nie działa w izolacji, lecz komunikuje się z innymi aplikacjami, serwisami lub platformami, pobierając dane, wysyłając informacje lub synchronizując stan systemu. Mechanizmem umożliwiającym taką współpracę są interfejsy programistyczne aplikacji, czyli API (Application Programming Interface).

Zewnętrzne API najczęściej udostępniane są w formie usług sieciowych, do których dostęp odbywa się za pośrednictwem protokołu HTTP. Aplikacja kliencka lub serwerowa wysyła żądanie (np. GET lub POST), a następnie odbiera odpowiedź zawierającą dane – najczęściej w formacie JSON lub XML. Poprawna integracja wymaga nie tylko umiejętności wysłania żądania, lecz także interpretacji odpowiedzi, obsługi błędów, limitów zapytań oraz ewentualnych mechanizmów autoryzacji.

W praktyce integracja z zewnętrznymi systemami może dotyczyć bardzo różnych obszarów, takich jak pobieranie danych pogodowych, map, kursów walut, informacji o użytkownikach, czy komunikacja w czasie rzeczywistym. Istotnym aspektem jest również asynchroniczność – aplikacja nie powinna blokować interfejsu użytkownika ani głównego wątku serwera podczas oczekiwania na odpowiedź z zewnętrznej usługi.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Cel laboratorium

Celem laboratorium jest nabycie umiejętności integrowania aplikacji internetowej z zewnętrznym API, poprawnego wysyłania żądań HTTP oraz przetwarzania odpowiedzi otrzymywanych z innych systemów. Student pozna podstawowe problemy związane z komunikacją z usługami zewnętrznymi oraz nauczy się obsługi sytuacji błędnych.

Instrukcja laboratoryjna

Ćwiczenie 1

Analiza zewnętrznego API i dokumentacji

1. Wybierz publicznie dostępne API (np. pogodowe, informacyjne lub demonstracyjne).
2. Upewnij się, że API udostępnia dane w formacie JSON i pozwala na wykonywanie żądań metodą GET.
3. Sprawdź adres bazowy API (np. `https://api.example.com`).
4. Zidentyfikuj endpoint, który umożliwia pobranie danych.
5. Sprawdź, jakie parametry są wymagane (np. identyfikator, nazwa miasta, klucz API).
6. Przeanalizuj strukturę odpowiedzi – jakie pola zawiera obiekt JSON.

Wynik: Student zna adres endpointu, wymagane parametry oraz strukturę danych zwracanych przez zewnętrzne API.

Ćwiczenie 2

Przygotowanie aplikacji do komunikacji z zewnętrznym API

1. Utwórz plik JavaScript odpowiedzialny za komunikację z API (np. `api-client.js`).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

2. Upewnij się, że plik jest poprawnie dołączony do aplikacji.
3. Do realizacji zapytań HTTP użyj mechanizmu `fetch`.
4. Zwróć uwagę, że `fetch` działa asynchronicznie i zwraca obietnicę (Promise).

Przykład przygotowania funkcji:

```
function fetchExternalData(url) {
	return fetch(url);
}
```

Wynik: Aplikacja jest przygotowana do wysyłania asynchronicznych żądań HTTP do zewnętrznych usług.

Ćwiczenie 3

Wysyłanie żądania GET do zewnętrznego API

Celem tego zadania jest pobranie danych z zewnętrznego systemu.

1. Zbudowanie pełnego adresu URL.
   - Połącz adres bazowy API z odpowiednim endpointem.
   - Dodaj wymagane parametry zapytania.
2. Wysłanie żądania.
   - Wywołaj funkcję fetch z przygotowanym adresem URL.
   - Sprawdź status odpowiedzi HTTP.

Przykład:

```
fetch('https://api.example.com/data')
	.then(response => response.json());
```

Wynik: Aplikacja wysyła poprawne żądanie GET i otrzymuje odpowiedź z zewnętrznego API.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Ćwiczenie 4

Przetwarzanie odpowiedzi z API

1. Użyj metody `json()` na obiekcie odpowiedzi.
2. Odbierz obiekt JavaScript reprezentujący dane.
3. Sprawdź, które pola odpowiedzi są istotne.
4. Wyodrębnij potrzebne informacje.

Przykład:

```
.then(data => {
      console.log(data);
});
```

Wynik: Dane zewnętrzne są poprawnie przetworzone i dostępne w postaci obiektów JavaScript.

Ćwiczenie 5

Obsługa błędów komunikacji z zewnętrzną usługą

1. Sprawdź, czy odpowiedź serwera ma status 200.
2. W przypadku błędu wyświetl komunikat użytkownikowi.
3. Dodaj blok `catch` do obsługi błędów.
4. Zarejestruj błąd w konsoli.

Przykład:

```
fetch(url)
      .then(res => {
            if (!res.ok) throw new Error('Błąd połączenia');
            return res.json();
      })
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
    .catch(err => console.error(err));
```

Wynik: Aplikacja poprawnie reaguje na błędy połączenia i nie przerywa działania.

Ćwiczenie 6

Prezentacja danych pobranych z zewnętrznego API

1. Utwórz kontener, w którym dane będą wyświetlane.
2. Na podstawie danych z API wygeneruj elementy HTML.
3. Wstaw je do dokumentu.

Przykład:

```
function renderData(data) {
    const el = document.getElementById('result');
    el.textContent = data.name;
}
```

Wynik: Dane z zewnętrznej usługi są widoczne w interfejsie aplikacji.

Ćwiczenia samodzielne

Ćwiczenie 1

Zintegruj aplikację z innym publicznym API i pobierz inny zestaw danych.

Ćwiczenie 2

Dodaj obsługę parametrów zapytania przekazywanych dynamicznie przez użytkownika.

Ćwiczenie 3

Zaimplementuj mechanizm ponawiania zapytania w przypadku błędu.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 4

Rozszerz interfejs o informację o czasie odpowiedzi zewnętrznej usługi.

Ćwiczenie 5

Zaimplementuj prosty mechanizm buforowania odpowiedzi.

Pytania pomocnicze

1. Czym jest zewnętrzne API i jakie problemy rozwiązuje?
2. Jakie są różnice pomiędzy komunikacją synchroniczną i asynchroniczną?
3. Dlaczego obsługa błędów jest istotna przy integracji z innymi systemami?
4. W jaki sposób można ograniczyć liczbę zapytań do zewnętrznego API?
5. Jakie zagrożenia mogą wynikać z integracji z usługami zewnętrznymi?

Podsumowanie

W ramach laboratorium student zapoznał się z procesem integracji aplikacji z zewnętrznymi usługami poprzez API oraz z zasadami wysyłania i obsługi żądań HTTP. Zrealizowane zadania pozwoliły na praktyczne zrozumienie asynchronicznej komunikacji oraz przetwarzania danych pochodzących z innych systemów. Zdobyta wiedza stanowi podstawę do tworzenia aplikacji współpracujących z wieloma niezależnymi usługami oraz do rozwiązywania realnych problemów integracyjnych w projektach informatycznych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

POLITECHNIKA LUBELSKA

Materiały do przedmiotu:

PROGRAMOWANIE APLIKACJI INTERNETOWYCH

Laboratorium nr 15

Autor:

dr hab. inż. Michał Wydra, prof. Uczelni

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Wdrożenie (deploy) aplikacji internetowej jest jednym z kluczowych etapów cyklu życia oprogramowania, ponieważ to właśnie na tym etapie aplikacja staje się dostępna dla rzeczywistych użytkowników. W przeciwieństwie do środowiska deweloperskiego, które jest zoptymalizowane pod kątem wygody pracy programisty, środowisko produkcyjne musi spełniać znacznie bardziej rygorystyczne wymagania. Do najważniejszych z nich należą stabilność działania, bezpieczeństwo danych, wydajność, odporność na błędy oraz możliwość łatwego utrzymania i dalszego rozwoju systemu.

Przygotowanie aplikacji do wdrożenia obejmuje szereg działań technicznych i organizacyjnych. Jednym z podstawowych zagadnień jest właściwa konfiguracja środowiska uruchomieniowego. Oznacza to m.in. oddzielenie konfiguracji od kodu źródłowego, tak aby wartości takie jak porty, adresy baz danych, klucze API czy sekrety kryptograficzne nie były zapisane bezpośrednio w plikach aplikacji. Zamiast tego stosuje się zmienne środowiskowe, które mogą być niezależnie ustawiane w różnych środowiskach (development, test, production).

Istotnym elementem przygotowania do wdrożenia jest również budowanie wersji produkcyjnej aplikacji frontendowej. W nowoczesnych aplikacjach typu SPA (Single Page Application) kod JavaScript, arkusze stylów i zasoby statyczne są w procesie buildowania optymalizowane, minifikowane i łączone w taki sposób, aby zmniejszyć rozmiar plików i skrócić czas ładowania aplikacji w przeglądarce użytkownika. Wersja produkcyjna frontendu różni się więc znacząco od wersji uruchamianej lokalnie w trybie deweloperskim.

Po stronie serwerowej równie ważne jest uruchamianie aplikacji w odpowiednim trybie pracy. W środowisku produkcyjnym backend powinien działać jako stabilny proces, zarządzany przez dedykowane narzędzia, które umożliwiają automatyczny restart aplikacji w przypadku awarii, zbieranie logów oraz monitorowanie jej stanu. Dodatkowo aplikacja serwerowa bardzo często nie jest wystawiana bezpośrednio do Internetu, lecz działa za pośrednictwem serwera pośredniczącego (reverse proxy), który odpowiada za obsługę ruchu sieciowego, przekierowania oraz szyfrowanie połączeń.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Całość procesu wdrożeniowego wymaga także testów po uruchomieniu aplikacji w nowym środowisku. Tak zwane testy „smoke test" pozwalają szybko zweryfikować, czy podstawowe funkcjonalności aplikacji działają poprawnie po wdrożeniu i czy konfiguracja została wykonana zgodnie z założeniami. Zrozumienie wszystkich tych aspektów jest niezbędne, aby aplikacja internetowa mogła działać poprawnie, bezpiecznie i stabilnie w warunkach produkcyjnych.

Cel laboratorium

Celem laboratorium jest przygotowanie aplikacji (frontend + backend) do pracy w środowisku produkcyjnym poprzez zbudowanie wersji produkcyjnej, rozdzielenie konfiguracji od kodu, uruchomienie serwera w trybie produkcyjnym oraz sprawdzenie poprawności działania po wdrożeniu.

Instrukcja laboratoryjna

Ćwiczenie 1

Przygotowanie struktury projektu i rozdzielenie konfiguracji od kodu

1. Upewnij się, że projekt ma czytelny podział:
   - `server/` (backend)
   - `client/` (frontend)
   - `README.md` (instrukcje uruchomienia)
2. Sprawdź, czy w repozytorium nie znajdują się pliki z hasłami/kluczami.
3. W katalogu `server/` utwórz plik `.env`.
4. Dodaj podstawowe ustawienia (przykład):

```
PORT=8000

NODE_ENV=development

DB_HOST=localhost
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
DB_USER=app
DB_PASSWORD=app
DB_NAME=app_db
JWT_SECRET=change_me
```

5. Zainstaluj `dotenv`:

```
cd server
npm i dotenv
```

6. W pliku startowym (np. `server/index.js`) dodaj na samej górze:

```
import 'dotenv/config';
```

7. Zamień „twardo wpisane" wartości na odczyt z `process.env`:

```
const PORT = process.env.PORT || 8000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

8. Ustal zasadę: *production nie korzysta z pliku `.env` w repozytorium*.
9. Dodaj do `.gitignore`:

```
.env
.env.*
```

10. Utwórz plik `.env.example` (do dokumentacji) – bez sekretów:

```
PORT=
NODE_ENV=
DB_HOST=
DB_USER=
DB_PASSWORD=
DB_NAME=
JWT_SECRET=
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

Wynik: Projekt ma czytelną strukturę, a konfiguracja jest oddzielona od kodu i gotowa do ustawienia na serwerze produkcyjnym bez edycji plików źródłowych.

Ćwiczenie 2

Konfiguracja trybów uruchamiania backendu (dev vs production)

1. Otwórz `server/package.json`.
2. Dodaj/zmodyfikuj sekcję `scripts`:

```
{
   "scripts": {
        "dev": "NODE_ENV=development node index.js",
        "start": "NODE_ENV=production node index.js"
   }
}
```

   Jeśli pracujesz na Windows i nie masz ustawionej obsługi `NODE_ENV=...`, użyj pakietu `cross-env`:

```
npm i -D cross-env
```

   A potem:

```
"dev": "cross-env NODE_ENV=development node index.js",
"start": "cross-env NODE_ENV=production node index.js"
```

3. Dodaj prosty logger tylko w dev (przykład):

```
if (process.env.NODE_ENV !== 'production') {
   console.log('Running in development mode');
}
```

4. Upewnij się, że w produkcji nie zwracasz „pełnych” stack trace’ów do klienta.
5. Na końcu definicji tras dodaj middleware:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
app.use((err, req, res, next) => {
   const status = err.status || 500;
   const payload = {
        message: err.message || 'Internal Server Error'
   };
   if (process.env.NODE_ENV !== 'production') {
        payload.stack = err.stack;
   }
   res.status(status).json(payload);
});
```

Wynik: Backend uruchamia się w dwóch trybach (dev i production) oraz ma podstawową, kontrolowaną obsługę błędów różniącą się zakresem informacji zwracanych do klienta.

Ćwiczenie 3

Zbudowanie wersji produkcyjnej frontendu (React SPA)

1. Sprawdź, czy frontend uruchamia się lokalnie – wejdź do `client/` i uruchom:

```
npm install
npm start
```

2. Potwierdź w przeglądarce, że aplikacja działa.
3. W React (typowo) używa się zmiennych zaczynających się od prefiksu `REACT_APP_`.
4. Utwórz plik `client/.env.development`:

```
REACT_APP_API_URL=http://localhost:8000
```

5. Utwórz plik `client/.env.production`:

```
REACT_APP_API_URL=https://twoja-domena.pl
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

6. W kodzie, w miejscu wywołań `fetch`, użyj:

```
const API_URL = process.env.REACT_APP_API_URL;
fetch(`${API_URL}/api/endpoint`);
```

7. W katalogu `client/` uruchom:

```
npm run build
```

8. Po wykonaniu polecenia sprawdź, czy pojawił się katalog `build/`.
9. Zainstaluj prosty serwer statyczny:

```
npm i -g serve
```

10. Uruchom:

```
serve -s build -l 3000
```

11. Wejdź na `http://localhost:3000` i sprawdź działanie.

Wynik: Frontend ma wersję produkcyjną w postaci katalogu `build/` i działa poprawnie po uruchomieniu jako statyczne pliki.

Ćwiczenie 4

Połączenie frontendu i backendu w jednym wdrożeniu

1. Po zbudowaniu frontendu skopiuj client/build do server/public.
2. W Express dodaj serwowanie plików statycznych:

```
import path from 'path';
import express from 'express';

const app = express();
const __dirname = path.resolve();

app.use(express.static(path.join(__dirname, 'public')));
```

```
// Wszystko co nie jest /api - kieruj do index.html (obsługa routingu SPA)
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
```

3. Upewnij się, że trasy API mają prefiks `/api`:

```
app.use('/api', apiRouter);
```

4. Upewnij się, że `app.get('*')` jest zdefiniowane **po** trasach `/api`.
5. Uruchom backend:

```
cd server
npm run start
```

6. Otwórz w przeglądarce:
   - stronę główną SPA (np. `http://localhost:8000/`)
   - przykładowy endpoint (np. `http://localhost:8000/api/health`)

Wynik: Jedno uruchomienie serwera zapewnia dostęp do frontendu i backendu: SPA działa, a endpointy API odpowiadają pod `/api/....`

Ćwiczenie 5

Uruchomienie produkcyjne z menedżerem procesów (PM2)

1. Zainstaluj PM2 Na serwerze (lub lokalnie do testu):

```
npm i -g pm2
```

2. W katalogu `server/` uruchom:

```
pm2 start index.js --name app-server
```

3. Sprawdź status:

```
pm2 status
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

4. Podejrzyj logi:

```
pm2 logs app-server
```

5. Ustaw automatyczny start po restarcie serwera – wykonaj:

```
pm2 startup
```

6. Wykonaj komendę, którą PM2 wypisze w terminalu.
7. Zapisz konfigurację:

```
pm2 save
```

Wynik: Backend działa jako proces zarządzany przez PM2, posiada logi i może być automatycznie uruchamiany po restarcie systemu.

Ćwiczenie 6

Wystawienie aplikacji przez reverse proxy (Nginx) i test wdrożenia

1. Przygotuj założenia sieciowe. Ustal:
   - port aplikacji Node (np. 8000)
   - domenę (np. `twoja-domena.pl`)
2. Upewnij się, że Node nie działa na porcie 80/443.
3. Utwórz plik konfiguracyjny (np. `/etc/nginx/sites-available/app.conf`).
4. Wstaw konfigurację:

```
server {
   listen 80;
   server_name twoja-domena.pl;

   location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_http_version 1.1;
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

```
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
        }
    }
```

5. Włącz konfigurację i zrestartuj Nginx:

```
    sudo ln -s /etc/nginx/sites-available/app.conf /etc/nginx/sites-enabled/
    sudo nginx -t
    sudo systemctl restart nginx
```

Smoke test po wdrożeniu (checklista)

1. Wejdź na stronę główną: czy ładuje się SPA?
2. Wykonaj zapytanie do API: czy `/api/health` zwraca odpowiedź?
3. Sprawdź błędy w konsoli przeglądarki.
4. Sprawdź logi serwera:
   - `pm2 logs app-server`
   - logi Nginx (jeśli włączone)

Wynik: Aplikacja jest dostępna spod domeny przez Nginx, a podstawowe testy po wdrożeniu potwierdzają poprawne działanie frontendu i API.

Ćwiczenia samodzielne

Ćwiczenie 1

Dodaj endpoint `/api/health`, który zwraca `{ "status": "ok", "env": "production" }` (w produkcji bez dodatkowych danych). Sprawdź działanie lokalnie i za Nginx.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Ćwiczenie 2

Dodaj w React ekran błędu „Brak połączenia z serwerem” wyświetlany, gdy `fetch` zwróci błąd sieciowy lub status 500.

Ćwiczenie 3

Zaimplementuj prosty logger do pliku (np. osobny plik logów dla błędów) oraz upewnij się, że w produkcji logi nie zawierają sekretów.

Ćwiczenie 4

Dodaj konfigurację `CORS` tak, aby w development frontend mógł wołać backend z innego portu, a w produkcji CORS był ograniczony do konkretnej domeny.

Ćwiczenie 5

Przygotuj wariant wdrożenia, w którym frontend jest serwowany jako statyczne pliki przez Nginx, a backend działa tylko pod `/api` (Nginx rozdziela ruch w `location /` i `location /api`).

Ćwiczenie 6

Dodaj mechanizm „graceful shutdown”: po sygnale zakończenia procesu serwer przestaje przyjmować nowe połączenia i kończy działanie bez zrywania aktywnych żądań.

Pytania pomocnicze

1. Jakie są najważniejsze różnice między uruchomieniem aplikacji w trybie development i production?

2. Dlaczego konfiguracja powinna być przechowywana w zmiennych środowiskowych, a nie w kodzie?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

**Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00**

3. Po co buduje się wersję produkcyjną frontendu i dlaczego nie wdraża się serwera deweloperskiego?

4. Jaką rolę pełni reverse proxy (np. Nginx) i jakie problemy rozwiązuje w praktyce?

5. Jakie informacje o błędzie można ujawniać użytkownikowi w produkcji, a jakich nie należy ujawniać?

6. Co powinien obejmować „smoke test" po wdrożeniu i dlaczego warto go robić po każdej zmianie?

7. Jakie są skutki awarii procesu Node w produkcji i jak menedżer procesów minimalizuje ryzyko przestoju?

Podsumowanie

W ramach laboratorium student przygotował aplikację do wdrożenia poprzez rozdzielenie konfiguracji od kodu, zbudowanie wersji produkcyjnej frontendu oraz uruchomienie backendu w stabilnym trybie produkcyjnym. Następnie aplikacja została wystawiona do sieci z użyciem reverse proxy, a poprawność działania potwierdzono testami po wdrożeniu. Po realizacji ćwiczenia student powinien rozumieć, które elementy różnią produkcję od środowiska deweloperskiego, jak bezpiecznie zarządzać konfiguracją oraz jak diagnozować problemy na podstawie logów. Warto zastanowić się, które kroki wdrożeniowe były najbardziej podatne na pomyłki i jak można je zautomatyzować (np. skryptami lub pipeline CI/CD).

Literatura

[1] Duckett, Jon i in. HTML i CSS : zaprojektuj i zbuduj witrynę WWW. Gliwice: Helion, 2014.

[2] Mikowski, Michael S i in. Single Page Web Applications : programowanie aplikacji internetowych z JavaScript. Gliwice: Wyd. Helion, 2015.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

[3] Kozieł, Grzegorz i in. Bezpieczne aplikacje internetowe w PHP. Lublin: Wyd. Politechniki Lubelskiej, 2022.

[4] Dickey, Jeff i in. Nowoczesne aplikacje internetowe : MongoDB, Express, AngularJS, Node.js. Gliwice: Helion, 2016.

[5] Ater, Tal i in. Progresywne aplikacje webowe : potęga aplikacji natywnych w przeglądarce. Warszawa: APN Promise, 2018.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach:
FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS)
"POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00