



Zintegrowany
Program
Rozwoju
Politechniki
Lubelskiej -
część druga

**POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI
I INFORMATYKI**

**KIERUNEK STUDIÓW
INFORMATYKA**

*MATERIAŁY DO ZAJĘĆ
LABORATORYJNYCH*

Programowanie aplikacji internetowych

Autor:
dr Beata Pańczyk

Lublin 2021



**Fundusze
Europejskie**
Wiedza Edukacja Rozwój



**Rzeczpospolita
Polska**

Unia Europejska
Europejski Fundusz Społeczny



INFORMACJA O PRZEDMIOCIE

Cele przedmiotu:

Cel 1. Zapoznanie studentów ze specyfiką programowania aplikacji internetowej współpracującej z bazą danych z uwzględnieniem aspektów bezpieczeństwa na przykładzie PHP i MySQL.

Cel 2. Nabycie przez studentów umiejętności projektowania i programowania aplikacji internetowych w języku PHP współpracujących z bazą danych MySQL.

Efekty kształcenia w zakresie umiejętności:

Efekt 1. Student potrafi, zgodnie z zadaną specyfikacją, zaprojektować i zaprogramować aplikację internetową w języku PHP współpracującą z bazą danych, stosując poznane metody, techniki i narzędzia, pozyskując dodatkowo informacje z literatury i dokumentacji.

Efekt 2. Umie tworzyć bezpieczne oprogramowanie w środowisku PHP i MySQL stosując poznane algorytmy, metody, techniki i narzędzia oraz potrafi ocenić ich przydatność do rozwiązania danego problemu.

Efekt 3. Umie zaprojektować i zapisać w języku PHP proste algorytmy oraz potrafi zweryfikować poprawność ich działania.

Literatura do zajęć:

Literatura podstawowa

1. Nixon R., PHP, MySQL i JavaScript. Wprowadzenie. Helion, 2019.
2. <https://www.php.net/manual/en/index.php>, dokumentacja i podręcznik PHP.

Literatura uzupełniająca

1. <https://ph.pollub.pl/index.php/jcsi/article/view/145/65>, Jędrych S., Jędruszak B., Pańczyk B., Tworzenie aplikacji internetowych na platformie JEE i PHP – analiza porównawcza, Journal of Computer Sciences Institute, 2019, vol. 11, s. 86-90.
2. <https://ph.pollub.pl/index.php/iapgos/article/view/950/727>, Pańczyk B., Sławiński A., Technologie mapowania obiektowo-relacyjnego w aplikacjach PHP, Informatyka, Automatyka, Pomiary W Gospodarce I Ochronie Środowiska, 2015, nr 1, vol. 5, s. 29-32.

Metody i kryteria oceny:

Oceny cząstkowe:

- Ocena 1 Przygotowanie merytoryczne do zajęć laboratoryjnych na podstawie: wykładów, literatury, pytań kontrolnych do zajęć.
- Ocena 2 Realizacja i prezentacja wybranych zadań laboratoryjnych.

Ocena końcowa - zaliczenie przedmiotu:

- Pozytywne oceny cząstkowe.
- Prezentacja projektu końcowego aplikacji internetowej współpracującej z bazą danych.



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz Społeczny

Plan zajęć laboratoryjnych:

| | |
|-------------------------------|--|
| <u>Lab1.</u> | Środowisko programistyczne aplikacji internetowej – serwer Apache, MySQL i PHP. Podstawy tworzenia aplikacji PHP. |
| <u>Lab2.</u> | Obsługa formularzy w PHP. Tablice globalne. |
| <u>Lab3.</u> | Praca z plikami i katalogami |
| <u>Lab4.</u> | Filtryle i walidacja danych |
| <u>Lab5.</u> | Projektowanie oraz tworzenie klas dla zadanego problemu. Praca z plikami XML i JSON. |
| <u>Lab6.</u> | Praca z danymi w PHP i MySQL. Realizacja klasy do obsługi bazy danych w oparciu o rozszerzenie mysqli lub PDO. Stworzenie prostej aplikacji typu CRUD. |
| <u>Lab7.</u> | Implementacja mechanizmu sesji w aplikacjach PHP. Zastosowanie ciasteczek. |
| <u>Lab8.</u> | Obsługa rejestracji i logowania użytkownika |
| <u>Lab9.</u> | Implementacja mechanizmów zabezpieczeń przed popularnymi atakami na aplikacje internetowe |
| <u>Lab10.</u> | Modułowy szablon strony w oparciu o definicję klasy zarządzającej wyglądem strony |
| <u>Lab11.</u> | Asynchroniczna transmisja danych z serwera z wykorzystaniem PHP, AJAX i jQuery |
| <u>Lab12.</u> | Implementacja wzorca projektowego MVC w przykładowej aplikacji typu CRUD |
| <u>Lab13.</u> | Prezentacja projektów |



LABORATORIUM 1. ŚRODOWISKO PROGRAMISTYCZNE APLIKACJI INTERNETOWEJ – SERWER APACHE, MySQL I PHP. PODSTAWY TWORZENIA APLIKACJI PHP.

Cel laboratorium:

Celem zajęć jest przygotowanie środowiska programistycznego do tworzenia aplikacji internetowej w języku PHP (instalacja pakietu XAMPP z serwerem HTTP Apache i serwerem bazy danych MySQL) oraz poznanie podstawowych elementów programowania skryptów PHP.

Zakres tematyczny zajęć:

Wybór edytora i środowiska programistycznego do tworzenia aplikacji internetowych w języku PHP. Budowa prostego skryptu PHP z zastosowaniem podstawowych instrukcji.

Pytania kontrolne:

1. Co jest niezbędne do programowania aplikacji po stronie serwera?
2. Czym różni się programowanie aplikacji typu client-side od server-side?
3. Za pomocą jakiego znacznika osadza się skrypty PHP w kodzie dokumentu HTML?
4. Jakie są podstawowe typy danych w PHP? Czy PHP jest językiem silnie typowanym?

Zadanie 1.1. Przygotowanie środowiska programistycznego

Kolejne zadania z *Programowania aplikacji internetowych* będą realizowane na lokalnym serwerze HTTP **Apache**, który jest dostępny w pakiecie **XAMPP** w systemie operacyjnym Windows lub Linux.

1. Przygotuj środowisko programistyczne – zainstaluj pakiet XAMPP (<https://www.apachefriends.org/pl/download.html>) lub skorzystaj z innego serwera HTTP interpretującego skrypty PHP.
2. Wyszukaj folder pakietu (**xampp**) oraz przejrzyj zawartość podkatalogu **htdocs**. Jeśli korzystasz z innego pakietu lub samodzielnie konfigurujesz środowisko pracy – postępowanie jest analogiczne, tylko nazwa folderu na dokumenty może być inna).
3. Uruchom serwer Apache (w przypadku XAMPP możesz skorzystać z **XAMPP Control Panel – run Server**) (Rys. 1.1).
4. Sprawdź, czy serwer działa prawidłowo - w przeglądarce wpisz adres np. **http://localhost/** (ewentualnie będzie dodatkowo potrzebny numer portu, na którym działa serwer Apache np. **http://localhost:80/**). Port o numerze 80 jest domyślnym portem dla serwera Apache i nie ma konieczności go podawać. Domyślnie też zostanie uruchomiony skrypt **index.php** z folderu **htdocs** (Rys. 1.2).

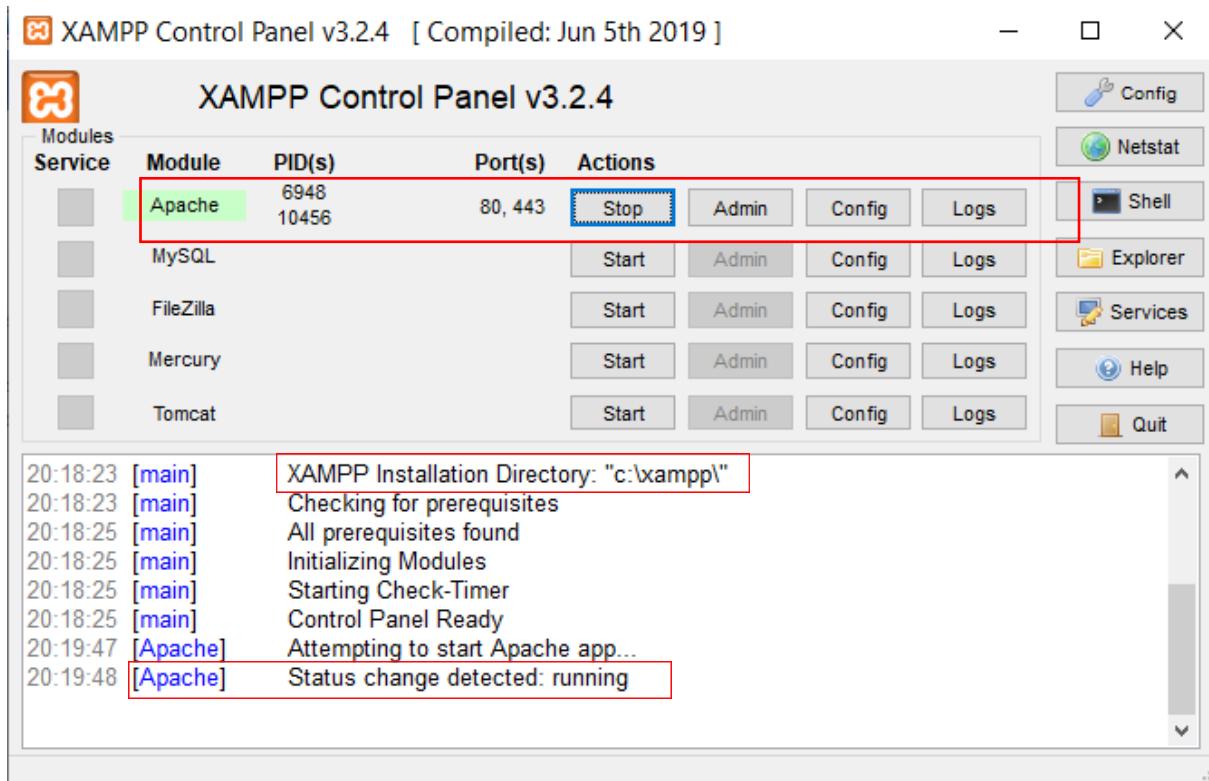


Fundusze Europejskie
Wiedza Edukacja Rozwój

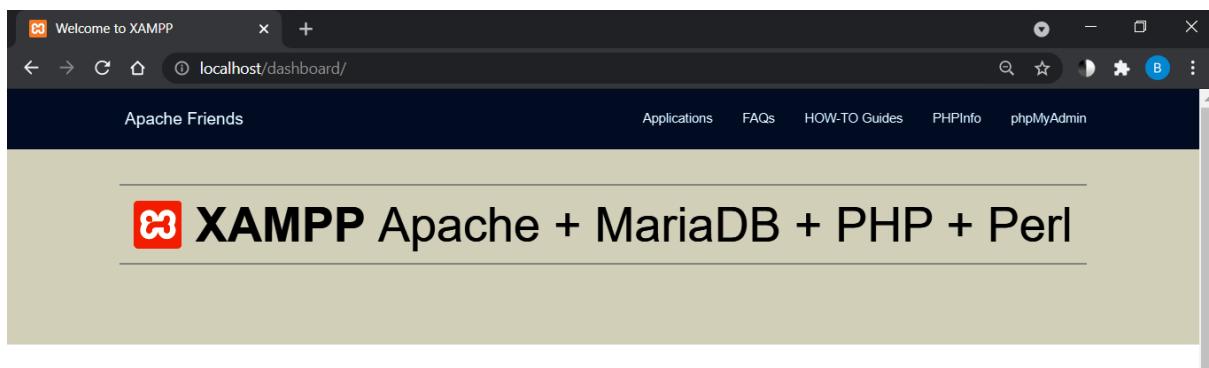


**Rzeczpospolita
Polska**





Rys. 1.1. Okienko Control Panel dla XAMPP



Welcome to XAMPP for Windows 8.0.2

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the FAQs section or check the HOW-TO Guides for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others. If you want have your XAMPP accessible from the internet, make sure you understand the implications and you checked the FAQs to learn how to protect your site. Alternatively you can use WAMP, MAMP or LAMP which are similar packages which are more suitable for production.

Start the XAMPP Control Panel to check the server status.

Rys. 1.2. Widok po uruchomieniu skryptu index.php na hoście lokalnym



Fundusze
Europejskie
Wiedza Edukacja Rozwój

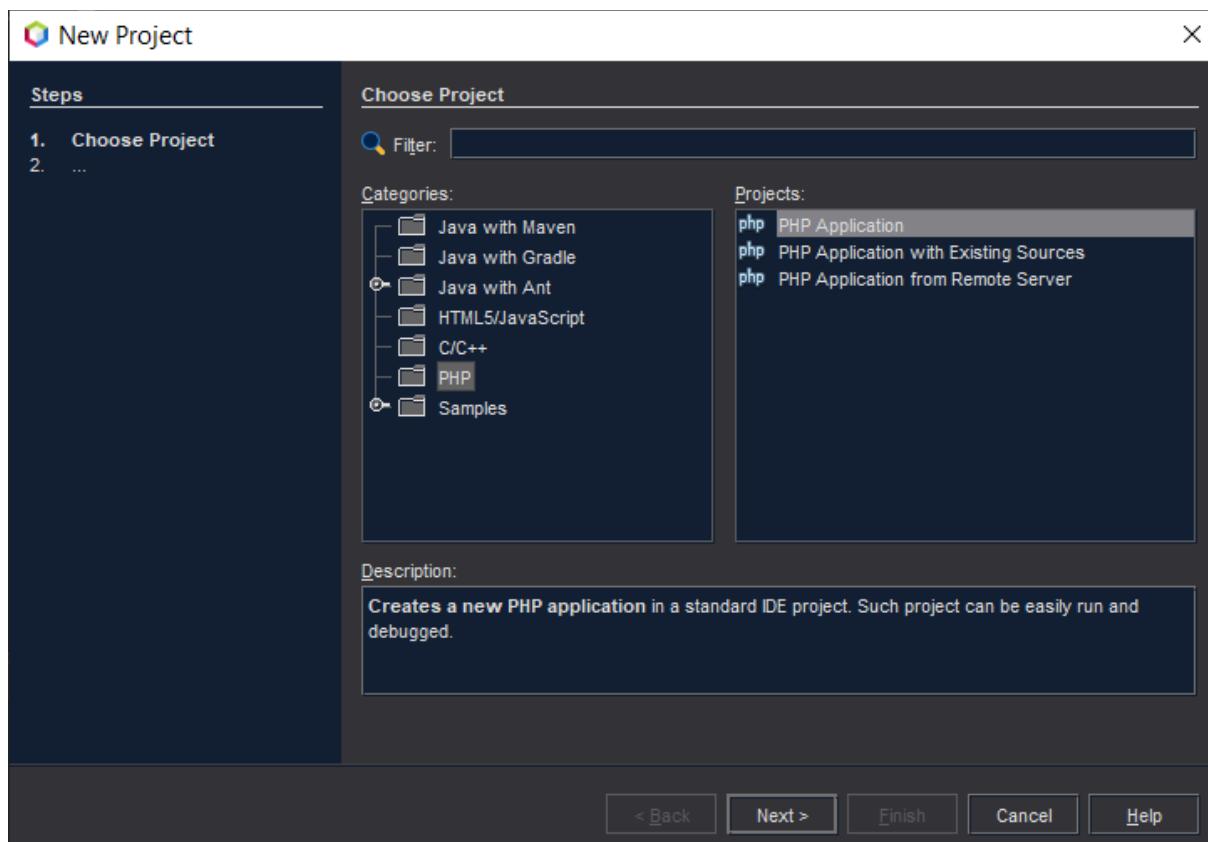


Rzeczpospolita
Polska



Zadanie 1.2. Utworzenie projektu PHP w NetBeans

1. Skrypty PHP można tworzyć w prostym edytorze tekstowym, jednak zdecydowanie wygodniej jest korzystać ze wsparcia środowiska programistycznego. Decyzja o wyborze środowiska jest sprawą indywidualną, ale w niniejszych materiałach przykłady demonstrowane są z zastosowaniem znanego już z poprzedniego semestru środowiska NetBeans.
2. Uruchom środowisko **NetBeans**, a następnie utwórz nowy projekt PHP (**File/ New Project**) jak pokazano na Rys. 1.3. Wskaż lokalizację folderu projektu (np. *c:\xampp\htdocs\phpai* jak na Rys. 1.4), zwróć uwagę na domyślnie przydzielony URL dla tworzonego projektu (np. <http://localhost/phpai/> jak na Rys. 1.5). Plik **index.php** jest domyślnie skryptem startowym projektu PHP (Rys. 1.6). W celu zmiany pliku startowego lub innych ustawień domyślnych, otwórz właściwości projektu (po wskazaniu myszką nazwy projektu, pod prawym klawiszem myszy dostępne jest menu kontekstowe z opcją wyboru jego właściwości - **Properties**). Przejrzyj dostępne kategorie np. **Run Configuration** (Rys. 1.7).



Rys. 1.3. Tworzenie nowego projektu PHP w NetBeans



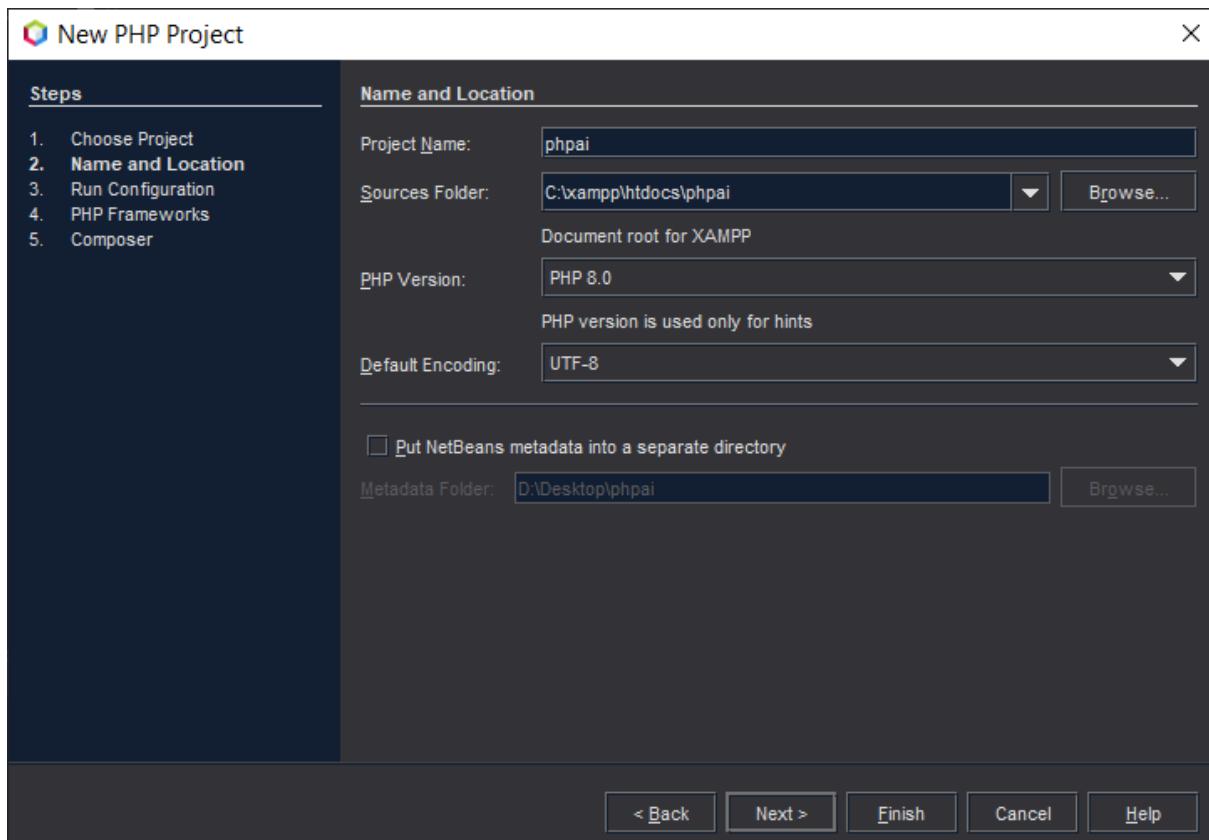
Fundusze
Europejskie
Wiedza Edukacja Rozwój



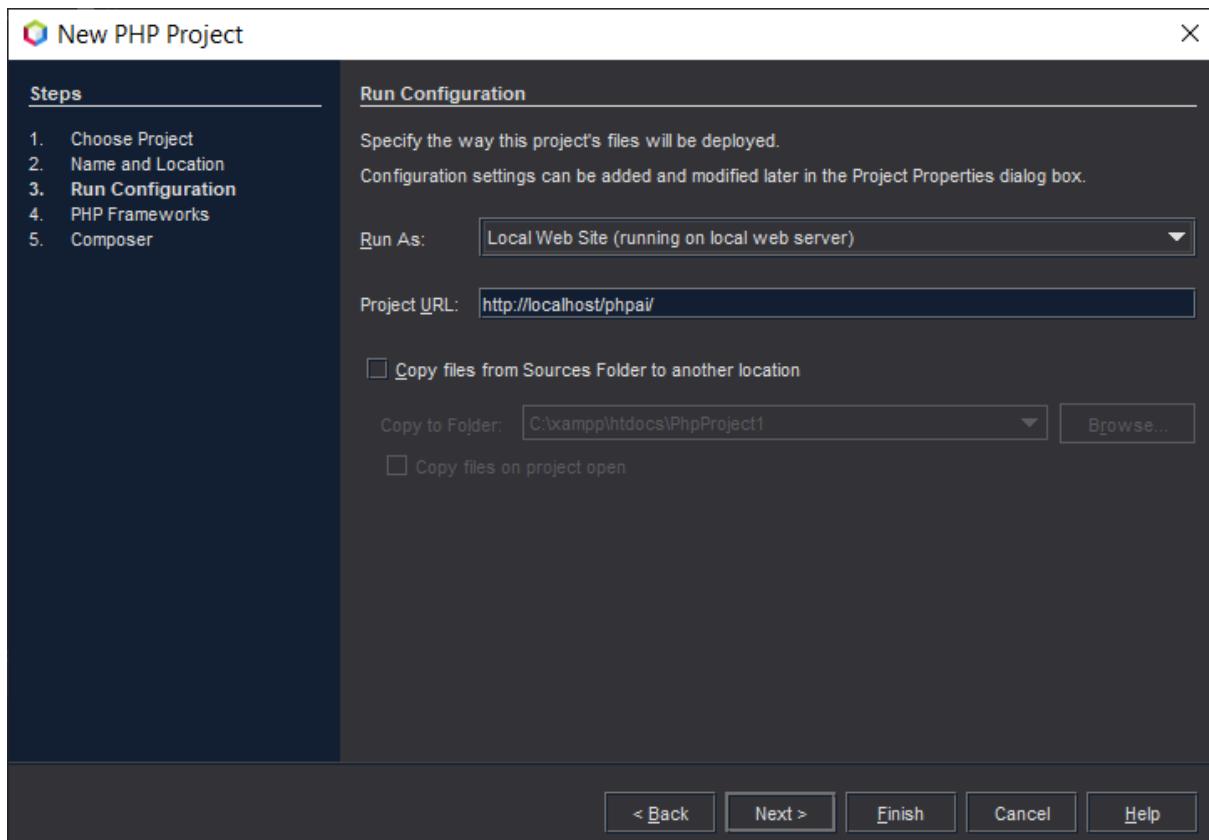
Rzeczpospolita
Polska



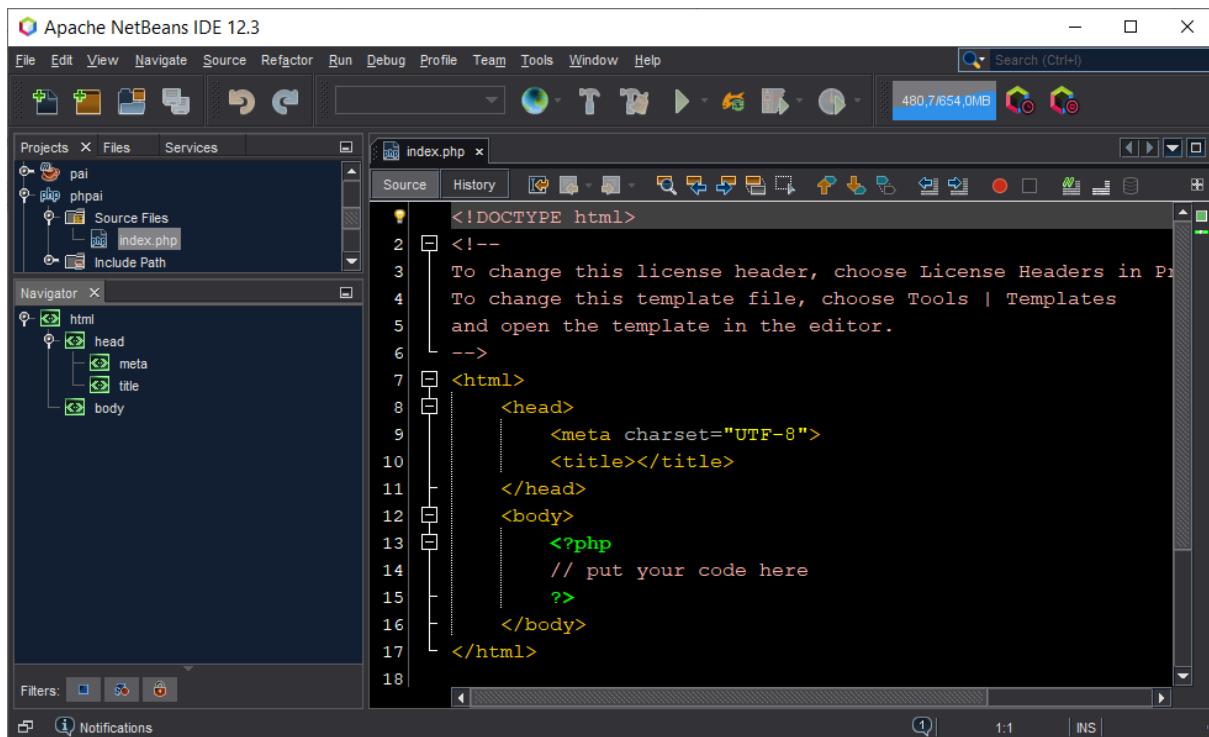
Unia Europejska
Europejski Fundusz Społeczny



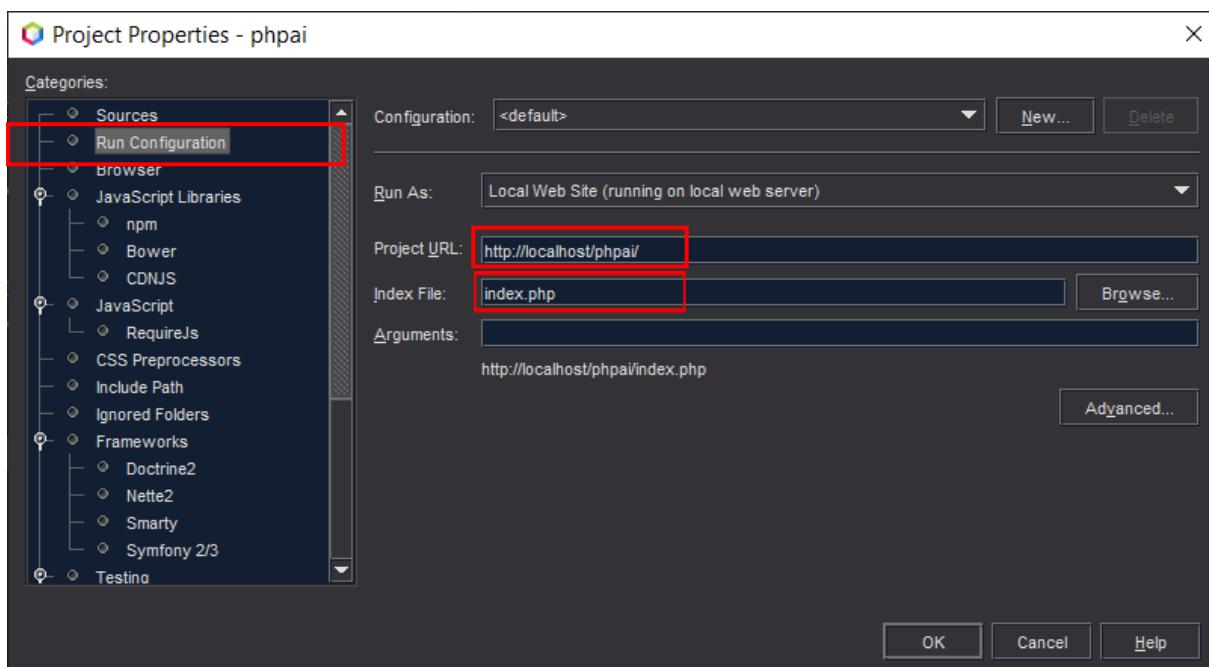
Rys.1.4. Nadanie nazwy i lokalizacji dla tworzonego projektu



Rys.1.5. Określenie adresu URL projektu

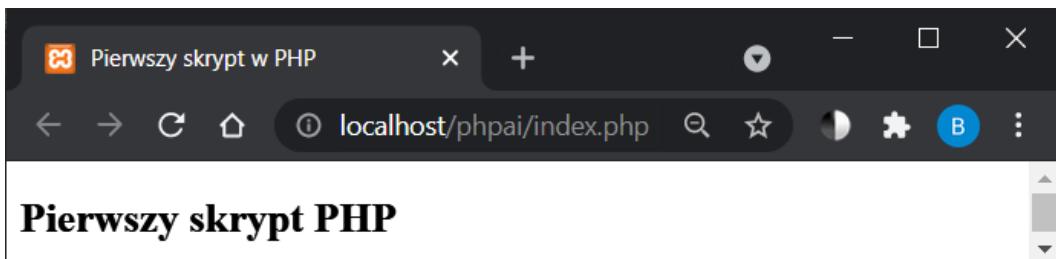


Rys. 1.6. Widok nowego projektu z plikiem startowym index.php w NetBeans



Rys. 1.7. Okienko Project Properties z właściwościami projektu PHP

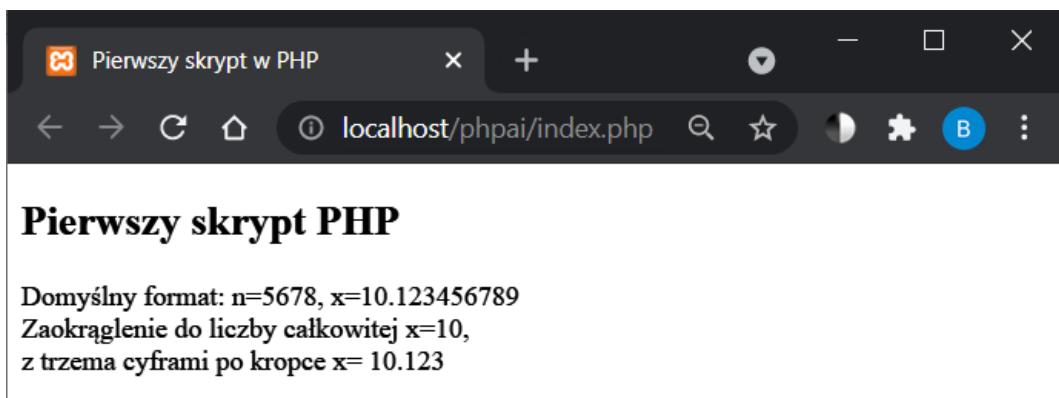
3. Uzupełnij skrypt **index.php** polecienniem wyświetlania komunikatu na stronie:
`echo "<h2>Pierwszy skrypt PHP</h2>"`. Dodaj też treść w elemencie `<title>`.
4. Za pomocą opcji **Run/Set Main Project** ustaw jako aktualny, właśnie utworzony nowy projekt PHP i go uruchom (**Run/Run Main Project** (lub skorzystaj z ikony zielonej strzałki). Efekt powinien być taki jak na Rys. 1.8. Zwróć uwagę na URL widoczny w pasku adresowym przeglądarki.



Rys. 1.8. Widok strony w przeglądarce po uruchomieniu pierwszego skryptu

Następnie w pliku *index.php* zdefiniuj zmienne, np. `$n=4567, $x=10.123456789` i wyświetl ich wartości na stronie jak na Rys. 1.9 za pomocą jednego polecenia *echo*:

- z wykorzystaniem operatora łączenia łańcuchów (znak kropki);
- bez stosowania operatora kropki;
- przy wyświetlaniu wartości bez stosowania operatora kropki sprawdź różnicę wyświetlania łańcucha ujętego w znaki " " i ' ';
- skorzystaj z funkcji *printf()* i zastosuj odpowiednie specyfikacje formatu dla wyświetlanych wartości (Rys. 1.9). Sprawdź parametry funkcji *printf()* w manualu PHP: <https://www.php.net/manual/en/index.php>



Rys. 1.9. Przykładowe wyniki realizacji punktów a-d.

Sprawdź kod źródłowy strony w przeglądarce (*Ctrl+U*). Czy widać, które fragmenty kodu zostały wygenerowane przez fragmenty kodu PHP?

Zadanie 1.3. Galeria w PHP

Wykorzystaj skrypt PHP do utworzenia galerii zdjęć za pomocą odpowiedniej iteracji. Zauważ, że pliki z miniaturkami (dostępne na platformie moodle) nazywają się odpowiednio *obraz1.jpg*.

- Do utworzonego projektu dodaj nowy plik *galeria* (*File->New File->PHP Web Page*) oraz folder *obrazki* z miniaturkami zdjęć.
- Kod HTML tworzący tablicę z miniaturami wygeneruj za pomocą skryptu PHP (najpierw wstaw pojedynczy obraz za pomocą kodu PHP – *Listing 1.1*, potem skorzystaj z pętli). Ostatecznie efekt w przeglądarce powinien być identyczny jak w przypadku statycznego kodu HTML – podejrzyj kod źródłowy uzyskany w wyniku działania skryptu z galerią w przeglądarce (*Ctrl+U*).
- Zdefiniuj funkcję, której parametrem będzie liczba wierszy i kolumn tablicy z obrazkami i przetestuj działanie funkcji dla parametrów 3x3, 4x2 oraz 2x4 (Rys.

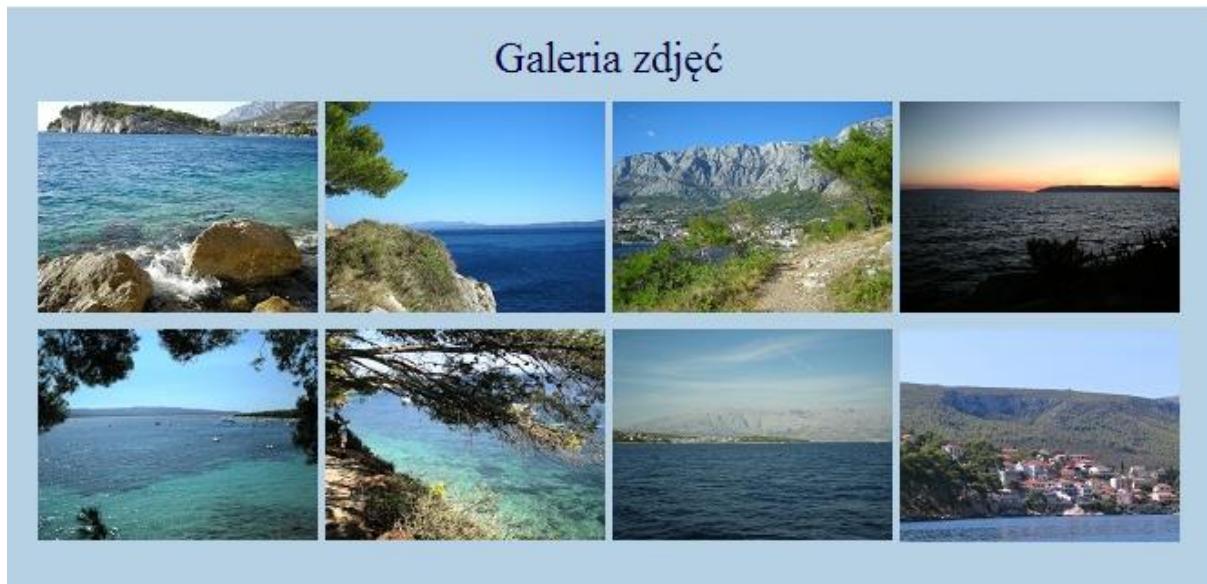
1.10). **Listing 1.2** przedstawia przykład definiowania funkcji bezpośrednio w skrypcie PHP.

Listing 1.1.

```
$nazwa='obraz1';
print("<img src='obrazki/$nazwa.JPG' alt='$nazwa' />" );
```

Listing 1.2. Schemat funkcji do wyświetlenia galerii zdjęć

```
<?php
    function galeria($rows, $cols)
    {
        //instrukcje
    }
    //wywołanie funkcji:
    galeria(3,3);
?>
```



Rys. 1.10. Przykładowy wynik działania skryptu galeria.php

Zadanie 1.4. Typy zmiennych w PHP

W kolejnym skrypcie (np. *typy.php*) utwórz zmienne o następujących wartościach:

- 1234
- 567.789
- 1
- 0
- true
- "0"
- "Typy w PHP"
- [1, 2, 3, 4]
- []
- ["zielony", "czerwony", "niebieski"]
- ["Agata", "Agatowska", 4.67, true]
- obiekt *DateTime* z bieżącą datą (<https://www.php.net/manual/en/class.datetime.php>).



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz Społeczny

- a) Wyświetl nazwy i wartości poszczególnych zmiennych (dla tablic skorzystaj z funkcji `count()` do uzyskania liczby elementów tablicy).
- b) Sprawdź na wybranych zmiennych działanie następujących funkcji: `is_bool()`, `is_int()`, `is_numeric()`, `is_string()`, `is_array()`, `is_object()`.
- c) Porównaj zmienne o wartościach **1** i **true** oraz **0** i **"0"**, za pomocą operatorów „**==**” oraz „**==**”. Wyświetl wynik porównania we wszystkich przypadkach.
- d) Dla tablic sprawdź jakie jest działanie pomocniczych funkcji `var_dump()` i `print_r()`, które wyświetlają informacje o zmiennych. Funkcje te są bardzo użyteczne na etapie testowania działania kodu skryptów.



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita Polska

Unia Europejska
Europejski Fundusz Społeczny



LABORATORIUM 2. OBSŁUGA FORMULARZY W PHP. TABLICE GLOBALNE.

Cel laboratorium:

Celem zajęć jest przygotowanie skryptu PHP do obsługi danych przesyłanych z formularza HTML z wykorzystaniem tablic globalnych.

Zakres tematyczny zajęć:

Przygotowanie formularza HTML. Opracowanie skryptu PHP, który odbierze przesłane w żądaniu HTTP dane z wypełnionego formularza. Zadaniem skryptu ma być również wygenerowanie strony wyświetlającej te dane.

Pytania kontrolne:

1. W jaki sposób można w skrypcie PHP odebrać parametry żądania przesłane metodą POST protokołu HTTP?
2. W jaki sposób można w skrypcie PHP odebrać parametry żądania przesłane metodą GET protokołu HTTP?
3. Co przechowują tablice globalne `$_GET`, `$_POST` i `$_REQUEST`?

Zadanie 2.1. Obsługa formularzy w PHP

- a) Do projektu z poprzedniego laboratorium dodaj nowy plik `formularz.html` z formularzem jak na Rys. 2.1. (kategoria **HTML File**) oraz nowy plik `odbierz.php` (**kategoria PHP Web Page**).
b) W skrypcie `odbierz.php` wyświetl dane przesłane z `formularz.html` (Rys. 2.2). Pamiętaj o ustawieniu odpowiednich atrybutów `action='odbierz.php'` i `method` w znaczniku `form` dokumentu `formularz.html` (Wykład 1, część 2). Przetestuj działanie metody GET i POST. Zastosuj narzędzia dostępne w przeglądarce w celu podglądu przesyłanych żądań z danymi z formularza. Zauważ, że w tablicy `$_REQUEST` znajdują się parametry przesyłane zarówno metodą GET, jak i POST. Zamiast korzystać z tablicy `$_REQUEST`, można również sięgnąć osobno do parametrów przesłanych metodą GET (tablica `$_GET`) czy POST (tablica `$_POST`).

UWAGA!

Skrypt, który odczytuje parametry przesłane w żądaniu HTTP, na początek powinien sprawdzać ich istnienie. Do tego celu skorzystaj z funkcji `isset()`, która sprawdza, czy parametr istnieje w odpowiedniej tablicy i czy jego wartość jest różna od `NULL`. Przy odczytywaniu danych pobranych z pól tekstowych pamiętaj o odpowiednim wykorzystaniu funkcji `trim()` i `htmlspecialchars()`. Fragment kodu z Listingu 2.1 przedstawia prawidłowe wykorzystanie niezbędnych funkcji.

Listing 2.1. Przykładowy kod skryptu odbierającego dane z żądania HTTP

```
<div>
    <h2>Dane odebrane z formularza:</h2>
    <?php
        if (isset($_REQUEST['nazw'])&&($_REQUEST['nazw']!="")) {
            $nazwisko = htmlspecialchars(trim($_REQUEST['nazw']));
        }
    </?php>
```



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska



Unia Europejska
Europejski Fundusz Społeczny

```
        echo "Nazwisko: $nazwisko <br />";  
    }  
    else echo "Nie wpisano nazwiska <br />";  
    //pozostałe instrukcje pobierające dane wysłane  
    //z formularza w postaci parametrów żądania  
    //...  
?>  
</div>
```

Formularz zamówienia

Nazwisko: Agatowska

Wiek: 21

Państwo: Polska

Adres e-mail: agatka@gmail.com

Zamawiam tutorial z języka:

PHP C/C++ Java

Sposób zapłaty:

eurocard visa przelew bankowy

Rys. 2.1. Wygląd formularza z pliku formularz.html

Dane odebrane z formularza

Nazwisko: Agatowska

Wiek: 21

Kraj: Polska

Email: agatka@gmail.com

Wybrano tutoriale: PHP Java

Sposób zapłaty: euro

[Powrót do formularza](#)

Rys. 2.2. Przykładowy efekt działania skryptu odbierz.php



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita
Polska

Unia Europejska
Europejski Fundusz Społeczny



- c) Przygotuj i przetestuj kolejny skrypt **odbierz2.php**, który pokaże nazwy i wartości wszystkich parametrów z globalnych tablic asocjacyjnych: **\$_GET**, **\$_POST** i **\$_REQUEST**. W tym celu wykorzystaj pętlę **foreach** postaci np.:
- ```
foreach($_REQUEST as $key=>$value) {
 echo "$key = $value
";
}
```
- d) Przetestuj ten skrypt dla formularza z punktu 1, kolejno ustawiając metodę przesyłania danych na **method="GET"**, potem zmieniając na **method="POST"**. Jaka jest różnica w działaniu skryptu dla obu metod?
- e) W celu szybkiego przejrzenia danych w tablicach globalnych – sprawdź na nich działanie funkcji **var\_dump()**.

### Zadanie 2.2. Skrypt formularz.php i odbierz3.php

Przygotuj kolejny skrypt **formularz.php**, który buduje taki sam formularz HTML jak w zadaniu 2.1, tyle, że z dodatkowymi przyciskami do wyboru kursów (Rys. 2.3).

W tym celu fragment z przyciskami typu checkbox wygeneruj za pomocą pętli w skrypcie PHP, stosując dla każdego przycisku taką samą wartość atrybutu **name="jezyki[]"**, ale inną wartość atrybutu **value** pobieraną z tablicy:

**\$jezyki = ["C", "CPP", "Java", "C#", "HTML", "CSS", "XML", "PHP", "JavaScript"];**  
Opracuj nowy skrypt **odbierz3.php**, który wyświetli dane o wybranych przyciskach (tutorialach) z nowego formularza za pomocą:

- pętli **foreach** (dla tablicy wartości **\$\_REQUEST['jezyki']**),
- funkcji **join()** lub **implode()** (np. **join(",",\$REQUEST['jezyki'])**).

Sprawdź ponownie działanie funkcji **var\_dump()** dla tablicy **\$\_REQUEST** i zwróć uwagę, że wartości dla niektórych kluczy (np. 'jezyki') są **tablica wartości typu String**. W skrypcie **odbierz3.php** dodaj kolejną pętlę **foreach**, która wyświetli wszystkie parametry tablicy **\$\_REQUEST** (nazwy i ich wartości), pamiętając, że niektóre wartości są tablicami (wykorzystaj funkcję **is\_array()**).

Warto też sprawdzić w manualu przykłady zastosowania funkcji **join()/implode()** oraz pętli **foreach**.

The screenshot shows a web browser window with the URL `localhost/phpai/formularz.php`. The page contains the following content:

**Zamawiam tutorial z języka:**

C  CPP  Java  C#  HTML  CSS  XML  PHP  JavaScript

**Sposób zapłaty:**

eurocard  visa  przelew bankowy

**Wyślij** **Anuluj**

Rys. 2.3. Formularz z dodatkowymi przyciskami do wyboru kursów



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



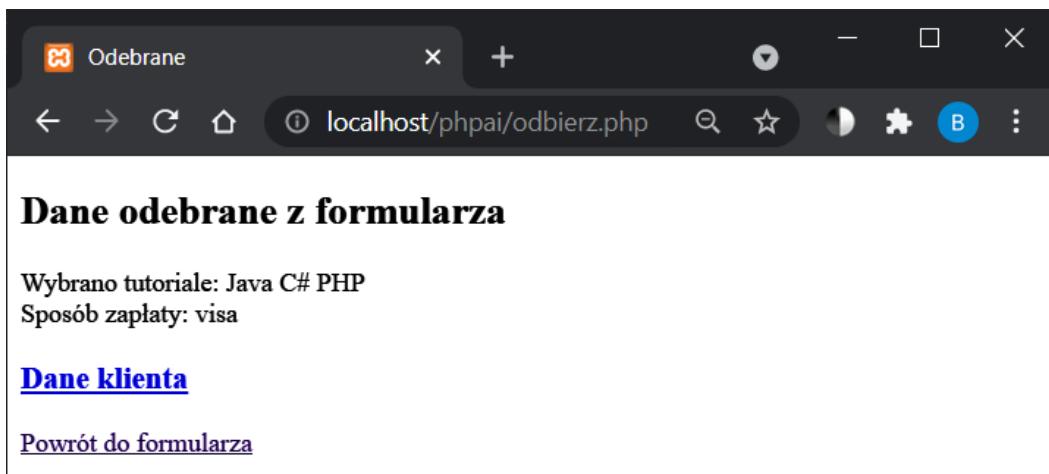
Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

### Zadanie 2.3. Przesyłanie danych pomiędzy stronami przez hiperłącza

Korzystając z przykładu wykładowego (Wykład 1, część 2) przygotuj skrypt **odbierz4.php**, który zamiast wyświetlać wszystkie dane z formularza jak na Rys. 2.2, wyświetli tylko zamówione produkty oraz sposób zapłaty i hiperłącze do danych zamawiającego obsługiwanych przez skrypt **klient.php** (Rys. 2.4). Dodatkowo zabezpiecz skrypt tak, aby w przypadku braku danych klienta pojawiał się odpowiedni komunikat zamiast hiperłącza.



Rys. 2.4. Efekt działania skryptu odbierz4.php

### Zadanie 2.4. Działanie funkcji htmlspecialchars()

Korzystając z przykładu wykładowego (Wykład 1, część 2), przetestuj działanie podanego tam skryptu, który jednocześnie realizuje wyświetlenie i obsługę formularza. Przetestuj działanie skryptu, wprowadzając do pola tekstowego kolejno:

- zwykły tekst,
- tekst z tagami HTML,
- tekst z kodem JavaScript.

Testy wykonaj bez zastosowania funkcji **htmlspecialchars()**, a następnie sprawdź, co się zmieni po jej wykorzystaniu.

## LABORATORIUM 3. CZEŚĆ 1. PRACA Z PLIKAMI I KATALOGAMI

### Cel laboratorium:

Celem zajęć jest poznanie funkcji PHP umożliwiających wykonywanie operacji na plikach tekstowych.

### Zakres tematyczny zajęć:

Zapoznanie się z funkcjami do obsługi plików. Dane odebrane z formularza będą zapisywane w plikach tekstowych z możliwością ich przeglądania. Do zadań wykorzystany zostanie lekko zmodyfikowany formularz przygotowany na poprzednim laboratorium.

### Pytania kontrolne:

1. Jakie znasz funkcje w PHP, umożliwiające otwarcie pliku?
2. Jakie są przykładowe metody do odczytu danych z pliku?
3. Jakie są przykładowe metody do zapisu danych do pliku?
4. W jaki sposób synchronizuje się dostęp do danych?

### Zadanie 3.1. Zapis danych z formularza do pliku tekstuowego

Do zadania wykorzystaj projekt utworzony na poprzednim laboratorium oraz przykłady z wykładu 2.

- a) Stwórz kopię skryptu **formularz.php** z poprzedniego laboratorium o nazwie **pliki.php** z dodatkowymi przyciskami typu **submit**. Każdy z przycisków powinien posiadać taką samą wartość atrybutu **name**, np. **name='submit'** i różne wartości atrybutu **value** (Rys. 3.1). Atrybut **action** dla formularza należy ustawić na **pliki.php** (ten sam skrypt wyświetla formularz i odbiera dane) (Listing 3.1).

Rys. 3.1. Formularz z dodatkowymi przyciskami typu submit (skrypt pliki.php)

### UWAGA!

Dane przesłane z formularza można zbierać po kolej, jak pokazano na Listingu 3.1 w funkcji **dodaj()**, ale można to zrobić bardziej elegancko korzystając z pętli **foreach** i odpowiedniej tablicy globalnej, np. **\$\_POST**. Wybierz bardziej zrozumiały dla siebie sposób.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój

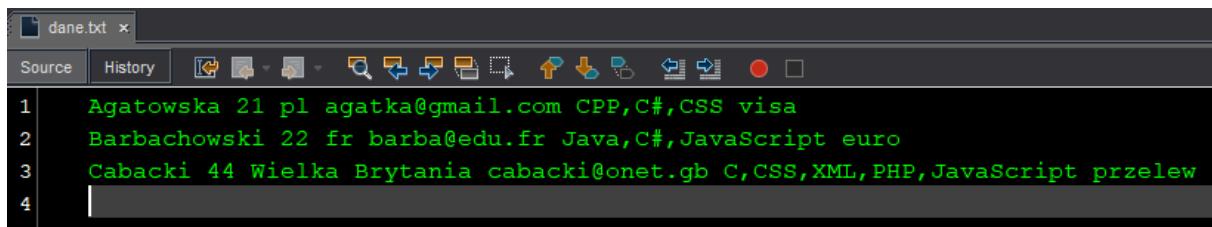


Rzeczpospolita  
Polska



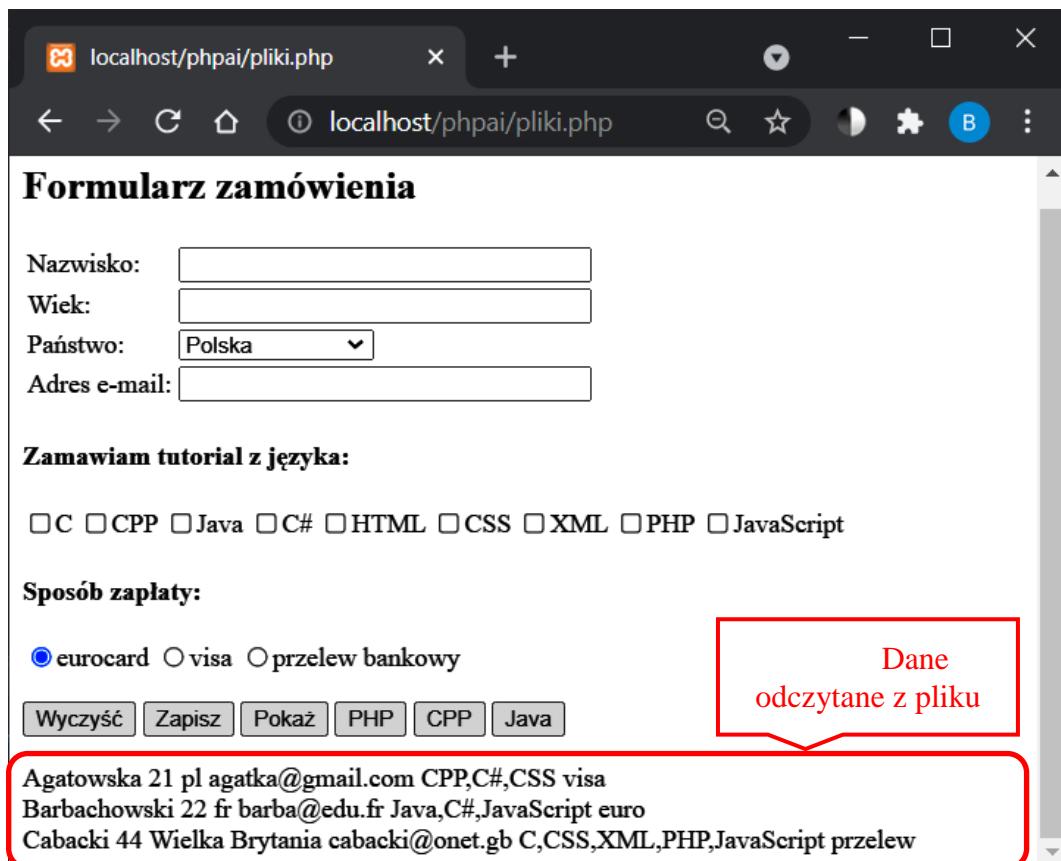
Unia Europejska  
Europejski Fundusz Społeczny

- b) Po wyborze przycisku **Zapisz** - za pomocą funkcji **dodaj()**, dane z formularza należy **dopisać** do pliku tekstowego **dane.txt** w lokalizacji domyślnej (na razie bez walidacji danych po stronie serwera). Sprawdź, gdzie zapisał się plik **dane.txt** i przejrzyj jego zawartość w dowolnym edytorze tekstowym. Najwygodniej jest dopisywać dane z każdego żądania w postaci kolejnego wiersza (Rys. 3.2). Taki format zapisu umożliwia łatwy dostęp do poszczególnych informacji. Zauważ, że poszczególne informacje rozdzielone są znakiem spacji, natomiast dane o zamawianych tutorialach scalane są w jeden łańcuch z separatorem **przecinka** (co będzie szczególnie ważne w przypadku zapisu danych do bazy danych).
- c) Po wyborze przycisku **Pokaż** - za pomocą funkcji **pokaz()** wyświetl wszystkie wiersze z pliku **dane.txt** (Rys.3.3).
- d) Po wyborze przycisków **Java**, **PHP**, **CPP** - za pomocą pomocniczej funkcji **pokaz\_zamowienie** - wyświetl tylko zamówienia na wskazane parametrem tutorialie.



```
1 Agatowska 21 pl agatka@gmail.com CPP,C#,CSS visa
2 Barbachowski 22 fr barba@edu.fr Java,C#,JavaScript euro
3 Cabacki 44 Wielka Brytania cabacki@onet.pl C,CSS,XML,PHP,JavaScript przelew
```

Rys. 3.2. Dane z formularza zapisane w pliku tekstowym dane.txt - widok w NetBeans



**Formularz zamówienia**

Nazwisko:

Wiek:

Państwo:

Adres e-mail:

**Zamawiam tutorial z języka:**

C  CPP  Java  C#  HTML  CSS  XML  PHP  JavaScript

**Sposób zapłaty:**

eurocard  visa  przelew bankowy

Dane odczytane z pliku

Agatowska 21 pl agatka@gmail.com CPP,C#,CSS visa  
Barbachowski 22 fr barba@edu.fr Java,C#,JavaScript euro  
Cabacki 44 Wielka Brytania cabacki@onet.pl C,CSS,XML,PHP,JavaScript przelew

Rys. 3.3. Przykładowy efekt działania przycisku Pokaż

Schemat skryptu **pliki.php** może wyglądać jak na Listingu 3.1.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



**Listing 3.1. Schemat skryptu pliki.php**

```
// poniższy kod dodaj na koniec dokumentu pliki.php (przed elementem
// zamykającym </body>

<?php
//Funkcje pomocnicze:
function dodaj() {
 $dane = "";
 if (isset($_REQUEST["nazw"])) {
 $dane .= htmlspecialchars($_REQUEST['nazw'])." ";
 }
 //zbierz pozostałe dane z formularza - dodając je do łańcucha $dane
 //zapisz łańcuch z danymi do pliku dane.txt w postaci wiersza np.:
 //Agatowska 21 Polska agatka@gmail.com PHP,CPP,Java Visa
}

function pokaz() {
 //odczytaj wszystkie dane z pliku i wyświetl wierszami...
}

function pokaz_zamowienie($tut) {
 //odczytaj dane z pliku i wyświetl tylko te wiersze,
 //w których zamówiono tutorial $tut (np. $tut="Java")
}

//Skrypt właściwy do obsługi akcji (żądań):
if (isset($_REQUEST["submit"])) { //jeśli kliknięto przycisk o name=submit
 $akcja = $_REQUEST["submit"]; //odczytaj jego value
 switch ($akcja) {
 case "Zapisz":dodaj();break;
 case "Pokaż":pokaz();break;
 case "Java":pokaz_zamowienie("Java");break;
 //pozostałe przypadki
 }
}
?>
```

- e) Wszystkie funkcje pomocnicze przenieś do pliku zewnętrznego o nazwie np. *funkcje.php* (w tym celu utwórz w projekcie dodatkowy plik, tym razem **tylko z kodem PHP**, wybierając kategorię: *New File->PHP File*) a następnie dodłącz go do skryptu *pliki.php* za pomocą np. *include\_once 'funkcje.php'*; przed instrukcjami do obsługi akcji.
- f) Przetestuj działanie programu z zewnętrznymi funkcjami. Efekt powinien być taki sam jak poprzednio.

**Zadanie 3.2. Zmiana lokalizacji pliku dane.txt**

Zapisz plik *dane.txt* nie w domyślnej lokalizacji (jak w zadaniu 3.1), ale spróbuj umieścić plik na serwerze poza ogólnie dostępnym folderem (*htdocs* - skorzystaj z wartości zmiennej serwerowej **DOCUMENT\_ROOT** w celu ustawienia względnej ścieżki do pliku (zobacz przykład *licznik* z wykładu 2).



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



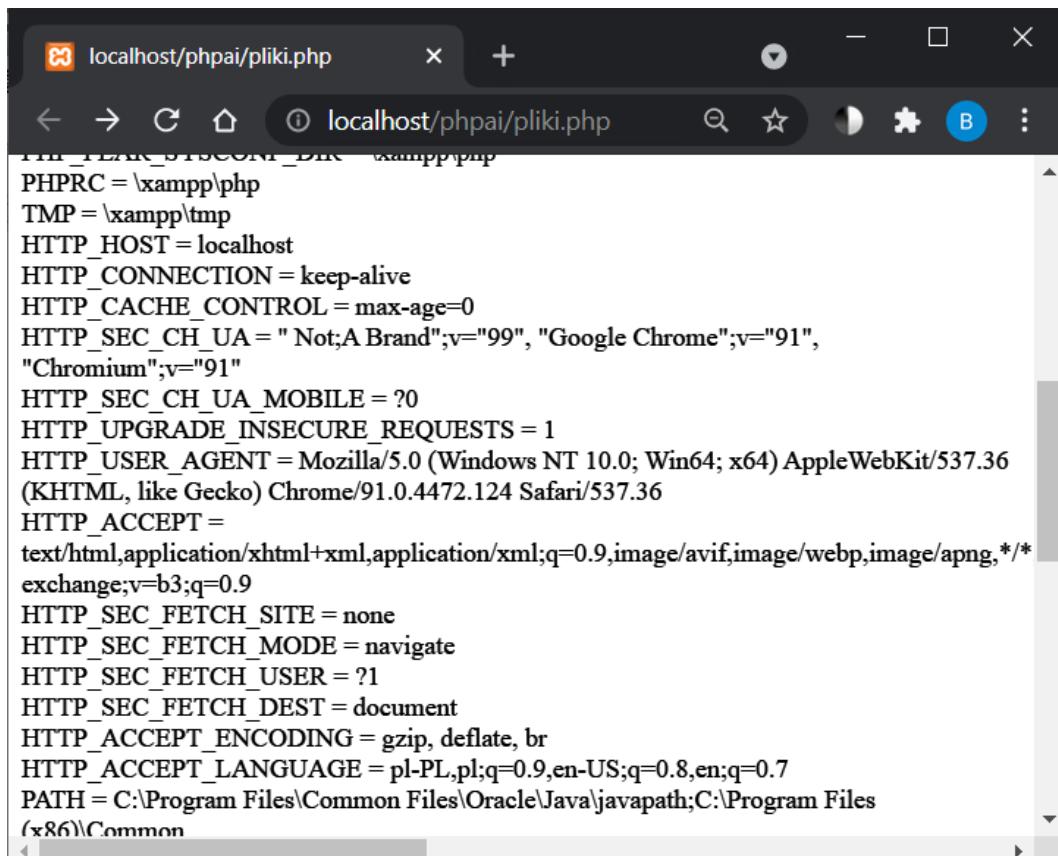
Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

### Zadanie 3.3. Tablica globalna \$\_SERVER

Na końcu skryptu dodaj polecenie wyświetlające wszystkie elementy tablicy globalnej **\$\_SERVER**. Jest to kolejna tablica asocjacyjna, która zawiera informacje udostępniane przez serwer związane z żądaniem HTTP i jego nagłówkami, dane serwera, lokalizacje plików (np. **HTTP\_ACCEPT**, **HTTP\_HOST**, **HTTP\_USERAGENT**, **SERVER\_SIGNATURE**, **SERVER\_NAME**, **SERVER\_PORT**, **DOCUMENT\_ROOT**) itp. Wpisy w tej tablicy są tworzone automatycznie przez serwer (Rys. 3.4).



The screenshot shows a browser window with the URL `localhost/phpai/pliki.php`. The page content displays the `$_SERVER` superglobal array, listing various environment variables and their values. Key entries include:

- `PHPRC = \xampp\php`
- `TMP = \xampp\tmp`
- `HTTP_HOST = localhost`
- `HTTP_CONNECTION = keep-alive`
- `HTTP_CACHE_CONTROL = max-age=0`
- `HTTP_SEC_CH_UA = "Not;A Brand";v="99", "Google Chrome";v="91", "Chromium";v="91"`
- `HTTP_SEC_CH_UA_MOBILE = ?0`
- `HTTP_UPGRADE_INSECURE_REQUESTS = 1`
- `HTTP_USER_AGENT = Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36`
- `HTTP_ACCEPT = text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9`
- `HTTP_SEC_FETCH_SITE = none`
- `HTTP_SEC_FETCH_MODE = navigate`
- `HTTP_SEC_FETCH_USER = ?1`
- `HTTP_SEC_FETCH_DEST = document`
- `HTTP_ACCEPT_ENCODING = gzip, deflate, br`
- `HTTP_ACCEPT_LANGUAGE = pl-PL,pl;q=0.9,en-US;q=0.8,en;q=0.7`
- `PATH = C:\Program Files\Common Files\Oracle\Java\javapath;C:\Program Files (x86)\Common`

Rys. 3.4. Przykładowe wartości tablicy `$_SERVER`



## LABORATORIUM 3. CZEŚĆ 2. PRACA Z PLIKAMI I KATALOGAMI

### Cel laboratorium:

Celem zajęć jest poznanie funkcji PHP umożliwiających wykonywanie operacji na plikach i katalogach.

### Zakres tematyczny zajęć:

Zapoznanie się z funkcjami do obsługi plików i katalogów na przykładzie skryptu umożliwiającego przesyłanie zdjęć i tworzenie ich miniatur.

### Pytania kontrolne:

1. Jakie znasz funkcje w PHP, umożliwiające wykonywanie operacji na plikach?
2. Jakie znasz funkcje w PHP, umożliwiające wykonywanie operacji na katalogach?

### Zadanie 3.4. Skrypt do przesyłania obrazów jpg i tworzenia miniatur zdjęć

Utwórz formularz HTML (np. *zdjecia.html*), który umożliwi użytkownikowi przesyłanie plików na serwer. Pamiętaj o ustawieniu odpowiednich atrybutów w elemencie **<form>** (**enctype="multipart/form-data"**) oraz odpowiedniego typu dla pola **<input>**, które umożliwi wybór pliku do przesłania na serwer (**type='file'**). Przykładowy kod formularza znajduje się na Listingu 3.4. Do pól typu numer dodaj atrybuty określające minimalne i maksymalne wartości określające wymiary tworzonej miniaturki w pikselach.

#### Listing 3.4.

```
<form method="post" action="zdjecia.php" enctype="multipart/form-data">
 Max. wys. <input type="number" name="wys">

 Max. szer. <input type="number" name="szer">

 Plik .jpg <input type="file" name="zdjecie">

 <input type="submit" value="Zapisz" name="zapisz">
</form>
```

PHP nie ogranicza się do tworzenia tylko danych wyjściowych typu HTML. Może być również używany do tworzenia i manipulowania plikami obrazów w różnych formatach graficznych, w tym GIF, PNG, JPEG. PHP może wysyłać strumienie obrazów bezpośrednio do przeglądarki. Aby to zadziałało, konieczna jest biblioteka funkcji graficznych GD. GD i PHP mogą również wymagać innych bibliotek, w zależności od tego, z jakim formatem obrazu pracujemy.

#### UWAGA!

Aby uzyskać dostęp do funkcji biblioteki GD, w pliku *php.ini* usuń komentarz w wierszu: **extension=gd** (lub dodaj taki wiersz, jeśli nie istnieje) a po każdej zmianie w pliku *php.ini* ponownie uruchom serwer Apache.

Przygotuj skrypt php odbierający pliki po stronie serwera, np. *zdjecia.php*.

W skrypcie:

1. Sprawdź, czy dane zostały przesłane (skorzystaj z funkcji `isset()`) i sprawdz, czy w żądaniu otrzymano już parametr o nazwie **pic** (przesyłany metodą **GET**). Parametr ten będzie potrzebny później, przy generowaniu linków do zdjęć.
2. Następnie skorzystaj z funkcji `is_uploaded_file()`, by sprawdzić, czy przesyłanie pliku na serwer zakończyło się powodzeniem. Jeśli tak, sprawdzamy, czy typ pliku jest zgodny z założeniami. Jeśli wszystko jest w porządku, zapisujemy wgrane zdjęcie w odpowiednim katalogu. Działania z punktów 1 i 2 realizuje kod:

```
if (isset($_POST['zapisz']) && $_POST['zapisz'] == 'Zapisz' &&
!isset($_GET['pic'])) {
 if (is_uploaded_file($_FILES['zdjecie']['tmp_name'])) {
 $typ = $_FILES['zdjecie']['type'];
 if ($typ === 'image/jpeg') {
 move_uploaded_file($_FILES['zdjecie']['tmp_name'], './'.
$_FILES['zdjecie']['name']);
 //kod będzie uzupełniany w kolejnych punktach
 }
}
```

3. Utwórz kopię zdjęcia, żeby wygodniej było pracować nad miniaturą oraz za pomocą funkcji **header** ustawić typ zawartości za pomocą nagłówka **Content-Type: image/jpeg**, by zasygnalizować przeglądarkę, że na wyjściu będzie odpowiedni typ danych. Pobierz informacje o wymiarach przesłanego zdjęcia i porównaj je z maksymalnymi wymiarami zalecanymi przez użytkownika, by ustawić poprawną skalę i wymiary miniatury. Działania te realizuje kolejny fragment kodu (do wstawienia po komentarzu w kodzie z punktu 2):

```
$link = $_FILES['zdjecie']['name'];
$random = uniqid('img_'); //wygenerowanie losowej wartości
$zdj = $random . '.jpg';
copy($link, './' . $zdj); //utworzenie kopii zdjęcia

list($width, $height) = getimagesize($zdj); //pobranie rozmiarów obrazu

$wys = $_POST['wys']; //wysokość preferowana przez użytkownika
$szer = $_POST['szer']; //szerokość preferowana przez użytkownika

$skalaWys = 1;
$skalaSzer = 1;
$skala = 1;
if ($width > $szer) $skalaSzer = $szer / $width;
if ($height > $wys) $skalaWys = $wys / $height;
if ($skalaWys <= $skalaSzer) $skala = $skalaWys;
else $skala = $skalaSzer;

//ustalenie rozmiarów miniaturki tworzonego zdjęcia:
$newH = $height * $skala;
$newW = $width * $skala;
```

4. Utwórz plik graficzny wypełniony kolorem czarnym o wymiarach miniatury (funkcja `imagecreatetruecolor()`), pobierz zawartość oryginalnego obrazu przy użyciu funkcji `imagecreatefromjpeg()`. Następnie skopiuj pobraną zawartość w całości do pliku



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



o wymiarach miniatury i zapisz go na serwerze korzystając z funkcji *imagejpeg()*. Po zakończeniu tych czynności usuń (funkcja *unlink()*) wcześniej stworzony czarny obraz oraz zawartość oryginalnego zdjęcia, a także kopię oryginalnego obrazu, stworzoną wcześniej. Operacje te realizuje kolejny fragment kodu:

```
header('Content-Type: image/jpeg');
$nowe = imagecreatetruecolor($newW, $newH); //czarny obraz
$obraz = imagecreatefromjpeg($zdj);
imagecopyresampled($nowe, $obraz, 0, 0, 0, 0,
 $newW, $newH, $width, $height);
imagejpeg($nowe, './mini-' . $link, 100);
echo "nowe=/mini-$link
";
imagedestroy($nowe);
imagedestroy($obraz);
unlink($zdj);
```

5. Miniatura zdjęcia przesłanego przez użytkownika została utworzona, więc pozostaje już tylko udostępnić link do niej i do oryginału. W tym celu wykorzystaj wcześniej wspomniany parametr o nazwie *pic* (przesyłany metodą **GET**). Ponieważ wcześniej zadeklarowano już w nagłówku, że typ odpowiedzi jest obrazem, więc przeglądarka nie wyświetli żadnego tekstu tylko obraz o nazwie przekazanej przez parametr *pic*. Po zakończeniu wykonywania skryptu tworzenia miniatury, przenieś użytkownika ponownie do tego samego pliku (**zdjecia.php**), ale z dodatkowym parametrem **zdjecia.php?pic=nazwa\_zdjecia**. W skrypcie dopisz linijkę sprawdzającą, czy metodą **GET** przesłano parametr *pic* i czy nie jest on pusty. Jeśli wszystko jest w porządku, na wyjście wyprowadzane będą linki do zdjęcia, które przesłał użytkownik i do jego miniatury, oraz link do powrotu na stronę formularza.

Fragment ...**else header('location: index.html');** odnosi się do wcześniejszego warunku **if**, który sprawdzał, czy typ przesłanego pliku był poprawny. Operacje te realizuje kod:

```
$dlugosc = strlen($link);
$dlugosc -= 4;
$link = substr($link, 0, $dlugosc);
echo "link=$link
";
header('location:zdjecia.php?pic=' . $link);
}
else {
 header('location:zdjecia.html');
}
```

Na koniec skryptu (powinny być tam już trzy klamry zamkające wcześniejsze instrukcje if, łącznie z pierwszą sprawdzającą parametry z POST) dodaj następną instrukcję warunkową, która sprawdza, czy przesłany został parametr *pic* (metodą GET):

```
if (isset($_GET['pic']) && !empty($_GET['pic']))
{
 echo 'Zdjęcie
';
 echo '
 Miniatura

';
 echo 'Powrót';
}
```



**UWAGA!**

Skontroluj, czy rozszerzenia utworzonych plików miniatur to .jpg czy .JPG - podejmij odpowiednie działania w celu normalizacji wielkości liter, stosując np. funkcję `strtoupper()`.

Struktura gotowego skryptu pokazana jest na rysunku 3.7, a ostateczny efekt działania przedstawiają rysunki 3.5 i 3.6.

Formularz do przesyłania zdjęć

Max. wys.

Max. szer.

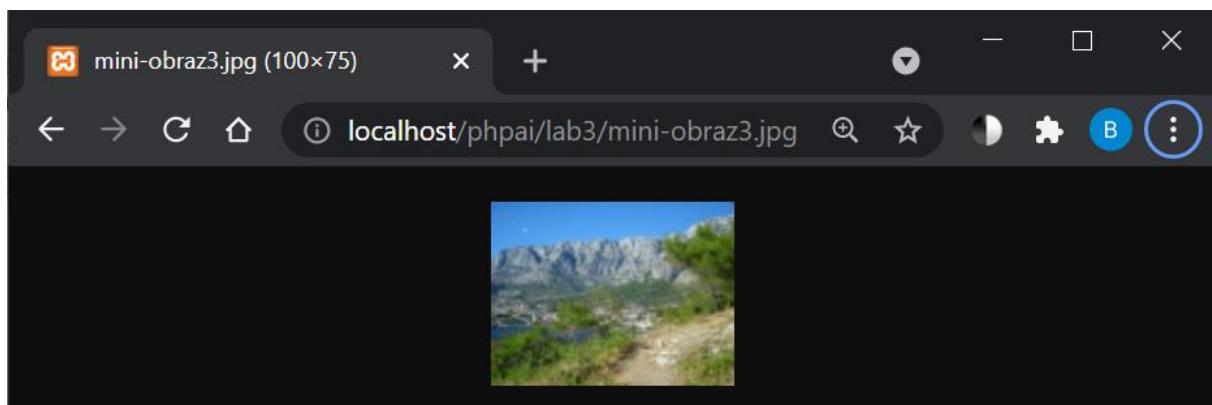
Plik jpg  obraz3.JPG

Zapisz

Rys. 3.5. Formularz do przesyłania zdjęć (plik `zdjecia.html`)

Zdjęcie  
Miniatura

Powrót



Rys. 3.6. Efekt działania skryptu i widok utworzonej miniatury dla przesłanego zdjęcia



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```

<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Miniaturki</title>
 </head>
 <body>
 <?php
 //odkomentować w php.ini: extension=gd
 if (isset($_POST['zapisz']) && $_POST['zapisz'] == 'Zapisz'
 && !isset($_GET['pic']))
 {
 if (is_uploaded_file($_FILES['zdjecie']['tmp_name']))
 [...50 lines...]
 }
 if (isset($_GET['pic']) && !empty($_GET['pic']))
 {
 echo 'zdjęcie
';
 echo 'Miniatuра

';
 echo 'Wróć';
 }
 ?>
 </body>
</html>

```

Rys. 3.7. Struktura gotowego projektu i schemat skryptu *zdjecia.php*

### Zadanie 3.5. Tworzenie galerii z wykorzystaniem skryptu z zad. 3.4.

1. Zmodyfikuj poprzedni skrypt *zdjecia.php*, tak aby miniaturki były zapisywane w folderze **miniatury**, a zdjęcia w folderze **zdjecia**. Najpierw utwórz te foldery w głównym katalogu projektu. Sprawdź, czy wszystko działa poprawnie.
2. Do skryptu *zdjecia.php* dodaj instrukcje wyświetlające miniatury wszystkich zdjęć, które już zostały przesłane na serwer (Rys. 3.8). W celu realizacji tego zadania skorzystaj z przykładów wykładu - praca z plikami i katalogami. Ponadto każda miniatuра ma stanowić jednocześnie hiperłącze do zdjęcia oryginalnych rozmiarów.

#### Galeria zdjęć



W galerii jest aktualnie 7 zdjęć

[Powrót do formularza dodawania zdjęć](#)

Rys. 3.8. Ostateczny efekt pracy skryptu *zdjecia.php* - każda miniatuра jest linkiem do oryginalnego zdjęcia



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## LABORATORIUM 4. FILTRY I WALIDACJA DANYCH

### Cel laboratorium:

Celem zajęć jest poznanie metod walidacji i filtrowania danych na przykładzie funkcji dostępnych w języku PHP.

### Zakres tematyczny zajęć:

Rozszerzenie zadania z laboratorium 3 poprzez wprowadzenie zabezpieczeń danych wejściowych za pomocą funkcji filtrującej *filter\_input* i różnych rodzajów filtrów typu VALIDATE i SANITIZE.

### Pytania kontrolne:

1. Jakie są parametry wejściowe funkcji *filter\_input* i co jest wynikiem jej działania?
2. Do czego służą filtry typu VALIDATE? Podaj przykłady takich filtrów.
3. Do czego służą filtry typu SANITIZE? Podaj przykłady takich filtrów.

### Zadanie 4.1. Przygotowanie środowiska programistycznego

1. Wszystkie definicje funkcji pomocniczych ze skryptu z zadania 3.1 (*pliki.php*) przenieś do pliku zewnętrznego *funkcje.php* (jeśli jeszcze nie zostało to wykonane w zadaniu 3.1) i dołącz definicje funkcji do skryptu *pliki.php* za pomocą funkcji *include\_once* lub *required\_once*. Sprawdź, czy wszystko działa poprawnie. Rysunki 4.1 i 4.2 przedstawiają schematy obu skryptów.
2. Następnie zmodyfikuj skrypt z zadania 3.1, tak aby zamiast funkcji *isset()* korzystał z funkcji *filter\_input()* (Listing 4.1). Sieganie bezpośrednio do danych w tablicach *\$\_GET*, *\$\_POST* czy *\$\_REQUEST* nie jest zalecane, dlatego dalej będziemy korzystać z bezpieczniejszego rozwiązania. Funkcja *filter\_input()* posłuży nam dalej również do celów walidacji danych wejściowych.
3. Stwórz nową wersję funkcji *dodaj()* (Listing 4.2) - przed zapisem do pliku funkcja powinna przeprowadzić walidację danych pobranych z pól formularza – zadania te wykona teraz nowa funkcja *walidacja()* (Listing 4.2), korzystająca z filtrów. Jeśli dane są błędne - wyświetl informację o błędach. Na Listingu 4.2 przedstawiono fragment filtrowania danych z formularza za pomocą funkcji *filter\_input\_array()* (przeanalizuj przykład z wykładu). Przykładowy efekt działania walidacji (i wynik wyświetlony przez funkcję *var\_dump(\$dane)* z Listingu 4.2) przedstawiają Rys. 4.3 i 4.4.

```
<?php
//Funkcje pomocnicze:
function dodaj() { ...23 lines }
function pokaz() { ...6 lines }
function pokaz_zamowienie($tut) { ...5 lines }
```

Rys. 4.1. Schemat skryptu *funkcje.php*



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title></title>
 </head>
 <body>
 <div>
 <h2>Formularz zamówienia</h2>
 <...49 lines />
 </div>
 <?php
 include_once "funkcje.php";
 //Skrypt właściwy do obsługi akcji:
 if (isset($_REQUEST["submit"])) {
 $akcja = $_REQUEST["submit"];
 switch ($akcja) {
 case "Zapisz":dodaj();break;
 case "Pokaż":pokaz();break;
 case "Java":pokaz_zamowienie("Java");break;
 //pozostałe przypadki
 }
 }
 ?>
</body>
</html>
```

Rys. 4.2. Schemat skryptu pliki.php z dołączonym skryptem funkcje.php

Listing 4.1. Zastosowanie funkcji filter\_input w skrypcie z zdania 3.1

```
<?php
 include_once "funkcje.php";
 if (filter_input(INPUT_POST, "submit")) {
 $akcja = filter_input(INPUT_POST, "submit");
 switch ($akcja) {
 case "Dodaj" : dodaj();break;
 case "Pokaż" : pokaz();break;
 //...
 }
 }
?>
```

Listing 4.2. Fragment kodu funkcji do walidacji danych z formularza

```
//Nowa funkcja
function walidacja() {
 $args = [
 'nazw' => ['filter' => FILTER_VALIDATE_REGEXP,
 'options' => ['regexp' => '/^([A-Z]{1}[a-zA-Ząźęłńśćżó-]{1,25})$/']],
 'kraj' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
];
}
```



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



```
'jezyki' => ['filter' => FILTER_SANITIZE_FULL_SPECIAL_CHARS,
 'flags' => FILTER_REQUIRE_ARRAY
]
//zdefiniuj pozostałe filtry
// . .
];

//przefiltruj dane z GET/POST zgodnie z ustawionymi w $args filtrami:
$dane = filter_input_array(INPUT_POST, $args);
//pokaż tablicę po przefiltrowaniu - sprawdź wyniki filtrowania:
var_dump($dane);

//Sprawdź czy dane w tablicy $dane nie zawierają błędów walidacji:
$errors = "";
foreach ($dane as $key => $val) {
 if ($val === false or $val === NULL) {
 $errors .= $key . " ";
 }
}

if ($errors === "") {
 //Dane poprawne - zapisz do pliku
 //wykorzystaj pomocniczą funkcję:
 dopliku("dane.txt", $dane);
} else {
 echo "
Nie poprawnie dane: " . $errors;
}
}

//nowa postać funkcji dodaj():
function dodaj(){
 echo "<h3>Dodawanie do pliku:</h3>";
 walidacja();
}

//nowa funkcja pomocnicza:
function dopliku($nazwaPliku, $tablicaDanych) {
 $dane = "";
 //zbierz wartości z tablicy danych (parametr $tablicaDanych
 //...
 $dane.=PHP_EOL; //dodaj koniec linii za pomocą stałej PHP
 //wykonaj operacje zapisu do pliku o zadanej nazwie:
 //
 echo "<p>Zapisano:
 $dane</p>";
}
```



**Formularz zamówienia**

Nazwisko:

Wiek:

Państwo:

Adres e-mail:

**Zamawiam tutorial z języka:**

C  CPP  Java  C#  HTML  CSS  XML  PHP  JavaScript

**Sposób zapłaty:**

eurocard  visa  przelew bankowy

**Dodawanie do pliku:**

```
array(6) { ["nazw"]=> bool(false) ["wiek"]=> bool(false) ["kraj"]=> string(2)
"pl" ["email"]=> bool(false) ["jezyki"]=> NULL ["zaplata"]=> string(4) "euro" }
Nie poprawnie dane: nazw wiek email jezyki
```

Rys. 4.3. Przykładowy efekt działania walidacji w przypadku niewprowadzenia danych do formularza zamówienia - `["nazw"]=> bool(false)` `["wiek"]=> bool(false)`  
`["email"]=> bool(false)` `["jezyki"]=> NULL`

**Dodawanie do pliku:**

```
array(6) { ["nazw"]=> string(10) "Kalinowska" ["wiek"]=> int(16) ["kraj"]=> string(2)
"pl" ["email"]=> string(17) "kalinka@gmail.com" ["jezyki"]=> array(2) { [0]=>
string(4) "HTML" [1]=> string(3) "CSS" } ["zaplata"]=> string(4) "visa" }
```

Zapisano:

Kalinowska 16 pl [kalinka@gmail.com](mailto:kalinka@gmail.com) HTML,CSS visa

Rys. 4.4. Przykładowy efekt działania walidacji w przypadku podania poprawnych danych



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



### Zadanie 4.2. Dodatkowe funkcjonalności skryptu pliki.php

W celach statystycznych do formularza dodaj kolejny przycisk '**Statystyki**', po wyborze którego na dole formularza należy wyświetlić informację o liczbie wszystkich zamówień w pliku oraz o liczbie zamówień pochodzących odpowiednio od osób poniżej 18 i powyżej 49 roku życia. Przykładowy efekt działania przedstawia Rys. 4.5.

The screenshot shows a form with three radio buttons for payment methods: eurocard (selected), visa, and przelew bankowy. Below the form are several buttons: Wyczysć, Zapisz, Pokaż, PHP, CPP, Java, and Statystyki. The text area displays statistical results:  
Liczba wszystkich zamówień: 6  
Liczba zamówień od osób w wieku < 18 lat: 2  
Liczba zamówień od osób w wieku >=50 lat: 1

Rys. 4.5. Przykładowy efekt działania skryptu *pliki.php* z wynikami statystyk

### Zadanie 4.3. Skrypt do obsługi ankiety

Utwórz nową stronę *ankieta.php* (PHP Web Page), w której należy umieścić formularz ankiety jak na rysunku 4.6. Przyciski typu *checkbox* wygeneruj za pomocą skryptu PHP, korzystając z tablicy:

```
$tech = ["C", "CPP", "Java", "C#", "Html", "CSS", "XML", "PHP", "JavaScript"];
```

Wyniki ankiety zapisuj w pliku *ankieta.txt* wierszami jak na Rys. 4.7. Po zapisie pokaż stronę z wynikami. Pojedyncze głosowanie powinno sprawdzać, na które technologie oddano głos i inkrementować w pliku odpowiednie liczniki. Możesz wykorzystać przykład ankiety z wykładu. Strona z wynikami powinna pokazać liczby głosów oddanych na każdy z języków po wielokrotnym głosowaniu (np. C -12, CPP - 10, ..., JavaScript - 21).

The form has a title 'Wybierz technologie, które znasz:'. It contains a list of checkboxes for various technologies: C, CPP, Java (checked), C#, Html, CSS, XML (checked), PHP, and JavaScript (checked). At the bottom is a 'Wyślij' button.

Rys. 4.6. Formularz ankiety

The screenshot shows a text editor window with the file 'ankieta.txt'. The content of the file is:  
Plik Edycja Format Widok Pomoc  
C:12  
CPP:10  
JavaScript:21

Rys. 4.7. Postać danych w pliku tekstowym *ankieta.txt*



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## LABORATORIUM 5. PROJEKTOWANIE ORAZ TWORZENIE KLAS DLA ZADANEGO PROBLEMU. PRACA Z PLIKAMI XML I JSON.

### Cel laboratorium:

Celem zajęć jest poznanie możliwości programowania skryptów PHP z wykorzystaniem paradygmatów programowania obiektowego na przykładzie aplikacji współpracującej z danymi z plików w formacie JSON i XML.

### Zakres tematyczny zajęć:

Przygotowanie skryptu z definicją klasy do zarządzania rejestracją użytkowników. Zastosowanie klasy w skrypcie realizującym zapis i odczyt obiektów klasy User w plikach w formacie JSON i XML.

### Pytania kontrolne:

1. Wymień składniki klasy i podaj przykład definicji podstawowej klasy w języku PHP.
2. Jakie specyfikatory dostępu do pól i metod klasy są stosowane w PHP? Jaki jest domyślny specyfikator?
3. Czy można w klasie PHP korzystać z przeciążania metod?
4. W jaki sposób w PHP można odwołać się w klasie pochodnej do konstruktora lub innej metody klasy bazowej?
5. W jaki sposób w PHP można odwołać się do składnika stałego i statycznego klasy:
  - a. w innej metodzie klasy?
  - b. z obiektu klasy?

### Zadanie 5.1. Definicja klasy User

Utwórz nowy projekt PHP (np. phplab5) a w nim nowy folder **klasy**. Do folderu dodaj nowy plik kategorii **PHP class**, w którym należy zdefiniować klasę **User** posiadającą następujące pola (Listing 5.1):

- *userName* - nazwę użytkownika,
- *passwd* - hasło,
- *fullName* - imię i nazwisko,
- *email* - adres e-mail,
- *date* - datę utworzenia konta (typu *DateTime*,  
<http://www.php.net/manual/pl/book.datetime.php>),
- *status* - status użytkownika (jako *int*, poszczególne wartości umieszczone w stałych klasy (STATUS\_ADMIN, STATUS\_USER).

Klasa (Listing 5.1) powinna posiadać konstruktor o sygnaturze (zwróć uwagę na podwójny znak podkreślenia poprzedzający specjalną nazwę metody konstruktora):

construct(\$userName, \$fullName, \$email, \$passwd )

Konstruktor ma za zadanie przypisać odpowiednie wartości do pól klasy, przy czym:

- a) *hasło* powinno być **zahasowane** za pomocą funkcji:

*password\_hash* (<http://php.net/manual/en/function.password-hash.php>),

- b) *status* użytkownika ustawić, korzystając z definicji pola stałego, na **STATUS\_USER** (wykły użytkownik),
- c) *datę* utworzenia ustawić na bieżącą datę z zastosowaniem klasy *DateTime* i jej metody *format* ('Y-m-d').

Do klasy dodaj także następujące metody:

- *show* - wyświetla poszczególne dane użytkownika korzystając z polecenia *echo*,
- *setUserName* - ustawia nazwę użytkownika,
- *getUserName* - zwraca nazwę użytkownika za pomocą wyrażenia *return*,
- pozostałe metody *get/set* - skorzystaj np. z autogeneratora środowiska NetBeans (*Insert code -> getter and setter*).

Listing 5.1 przedstawia fragment klasy *User*.

**Listing 5.1. Schemat definicji klasy User**

```
class User {
 const STATUS_USER = 1;
 const STATUS_ADMIN = 2;
 protected $userName;
 //pozostale pola klasy:
 //...

 //metody klasy:
 function __construct($userName, $fullName, $email, $passwd){
 //implementacja konstruktora
 $this->status=User::STATUS_USER;
 //nadać wartości pozostałym polom - zgodnie z parametrami
 //...
 }
 public function show() {
 //wyswietlic dane o obiekcie User:
 //...
 }
 //zdefiniowac pozostałe metody
}
```

W pliku *index.php* utwórz dwa obiekty klasy *User* i przetestuj działanie metod klasy:

- a) pokaż informacje o obu zdefiniowanych obiektach w postaci jak na Rys. 5.1,
- b) skorzystaj z metod *get/set* i dla jednego z obiektów zmień *userName* na 'admin' oraz zmodyfikuj *status* na **STATUS\_ADMIN** oraz ponownie wyświetl (metodą *show*) informacje o obiekcie. Sprawdź, czy rzeczywiście dane zostały zmienione.

Schemat skryptu *index.php* przedstawia Listing 5.2.

**Listing 5.2. Tworzenie obiektów klasy User w pliku index.php**

```
<?php
 include 'klasy/User.php';
 $user1 = new User ('kp', 'Kubus Puchatek',
 'kubus@stumilowylas.pl', 'nielubietygryska');
 $user1->show();
 //pozostałe instrukcje do uzupełnienia
?>
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



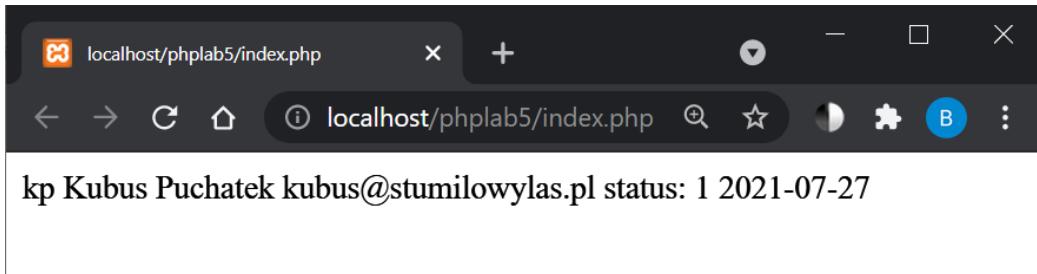
Rzeczypospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

**UWAGA!**

Pamiętaj, aby dołączyć plik z definicją klasy do skryptu, który z niej korzysta za pomocą funkcji **require\_once** lub **include\_once**.

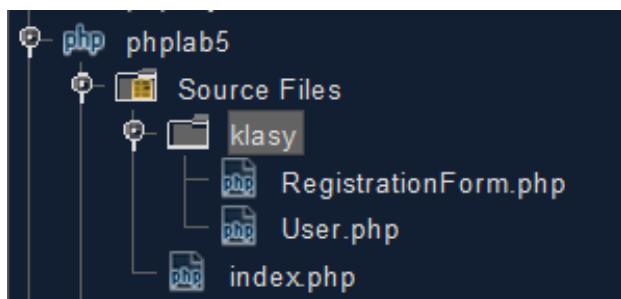


Rys. 5.1. Przykładowy wynik działania metody **show()** klasy **User**

**Zadanie 5.2. Klasa RegistrationForm - formularz rejestracji nowego użytkownika**

1. W folderze **klasy** zdefiniuj kolejną klasę **RegistrationForm** (plik **RegistrationForm.php** – Listing 5.3):
  - a) z polem chronionym **\$user** klasy **User**,
  - b) z konstruktorem bezparametrowym, który wyświetla formularz zawierający wszystkie niezbędne pola do rejestracji nowego użytkownika (Rys. 5.3).
  - c) z metodą **checkUser()**, która sprawdza, czy wszystkie niezbędne pola zostały podane prawidłowo, a jeśli tak, to tworzy nowy obiekt **\$user**, wypełniony wartościami pobranymi z wypełnionego formularza. W tym celu zastosuj funkcję walidacji i filtry podobnie jak w zadaniach z laboratorium 4. Jeśli dane nie są prawidłowe, metoda ustawia **\$this->user=NULL**. Jako wynik działania metoda zwraca obiekt **\$this->user**.
2. W pliku **index.php** (Listing 5.4) wyświetl formularz rejestracji a następnie zastosuj metodę **checkUser**. Metodawróci utworzony na podstawie danych obiekt **\$user**, gdy wszystkie dane są poprawne lub **NULL** w przeciwnym razie.
3. W przypadku kiedy rejestracja będzie poprawna - poniżej formularza (Rys. 5.3) wyświetl (za pomocą metody **show()**) dane nowego użytkownika. W przeciwnym razie wyświetl komunikat: „*Niepoprawne dane rejestracji*”.

Strukturę plików gotowego projektu przedstawia rysunek 5.2.



Rys. 5.2. Formularz rejestracji nowego użytkownika i efekt działania walidacji wprowadzonych danych



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Formularz rejestracji

Nazwa użytkownika:

Imię i nazwisko:

Hasło:

Email:

Rejestruj Anuluj

Poprawne dane rejestracji:  
kp Kubuś Puchatek kubus@stumilowy.las status: 1 2021-07-27

Rys. 5.3. Formularz rejestracji nowego użytkownika i efekt działania walidacji wprowadzonych danych

Listing 5.3. Fragment klasy RegistrationForm

```
<?php
class RegistrationForm {
 protected $user;
 function __construct(){ ?>
 <h3>Formularz rejestracji</h3><p>
 <form action="index.php" method="post">
 Nazwa użytkownika:
<input name="userName" />

 //dodaj pozostałe pola formularza
 </form></p>
 <?php
 }
 function checkUser(){ // podobnie jak metoda validate z lab4
 $args = [
 'userName' => ['filter' => FILTER_VALIDATE_REGEXP,
 'options' => ['regexp' => '/^[\0-9A-Za-zA-Ząźęłńśćżó_-]{2,25}$/']]
],
 // pozostałe warunki do walidacji
];
 //przefiltruj dane:
 $dane = filter_input_array(INPUT_POST, $args);
 //sprawdz czy są błędy walidacji $errors - jak w lab4
 ... // uzupełnij kod
 if ($errors === "") {
```



```
//Dane poprawne - utwórz obiekt user
$this->user=new User($dane['userName'], $dane['fullName'],
 $dane['email'],$dane['passwd']);

} else {
 echo "<p>Błędne dane:$errors</p>";
 $this->user = NULL;
}
return $this->user;
}
}
```

**Listing 5.4.** Schemat skryptu w pliku index.php

```
<?php
 include_once('klasy/User.php');
 include_once('klasy/RegistrationForm.php');
 $rf = new RegistrationForm(); //wyświetla formularz rejestracji
 if (filter_input(INPUT_POST, 'submit',
FILTER_SANITIZE_FULL_SPECIAL_CHARS)) {
 $user = $rf->checkUser(); //sprawdza poprawność danych
 if ($user === NULL)
 echo "<p>Niepoprawne dane rejestracji.</p>";
 else{
 echo "<p>Poprawne dane rejestracji:</p>";
 $user->show();
 }
 }
}
```

### Zadanie 5.3. Obsługa formatu JSON w PHP

#### UWAGA!

W PHP format JSON obsługują metody *json\_encode()* i *json\_decode()*. Zapoznaj się z ich składnią, korzystając z manuala PHP. Do pracy z plikami JSON są również użyteczne metody: *file\_get\_contents()* i *file\_put\_contents()* – sprawdź jak działają.

1. Do głównego folderu projektu dodaj nowy plik kategorii JSON: *users.json*, do którego skopiuj zawartość z Listingu 5.5.
2. Do klasy *User* dodaj metodę **statyczną** *getAllUsers()*, która będzie wyświetlała listę wszystkich zarejestrowanych użytkowników, wczytanych z pliku *users.json* (Listing 5.6). W pliku *index.php* dodaj wywołanie metody pokazującej wszystkich zarejestrowanych użytkowników (*User::getAllUsers()*) (Rys. 5.4.).
3. Do klasy *User* dodaj metodę *toArray()* i *save()*, która będzie zapisywała dane użytkownika do tablicy asocjacyjnej, zapisywanej docelowo w formacie JSON w pliku *users.json* (Listing 5.7).
4. W skrypcie *index.php*, po wyświetleniu prawidłowych danych rejestracji - dodatkowo zapisz do pliku nowo utworzonego użytkownika: *\$user->save("users.json")*. Sprawdź, czy nowy użytkownik został poprawnie dopisany do tablicy obiektów JSON w pliku (i w jakiej postaci jest zapisane jego hasło).



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



**Listing 5.5. Początkowa zawartość pliku users.json**

```
[
 { "userName" : "admin",
 "passwd" : "admin",
 "fullName" : "Administrator",
 "email" : "admin@gmail.com",
 "date" : "2018-09-10",
 "status" : 2
 },
 { "userName" : "kubus",
 "passwd" : "kubus",
 "fullName" : "Kubuś Puchatek",
 "email" : "kubus@stumilowylas.com",
 "date" : "2019-10-30",
 "status" : 1
 }
]
```

**Listing 5.6. Odczyt z pliku users.json – dodatkowa metoda w klasie User**

```
static function getAllUsers($plik){
 $tab = json_decode(file_get_contents($plik));
 //var_dump($tab);
 foreach ($tab as $val){
 echo "<p>".$val->userName." ".$val->fullName." ".$val->date."
</p>";
 }
}
```

**Listing 5.7. Metody toArray i save – dodatkowe metody klasy User**

```
function toArray(){
 $arr=[
 "userName" => $this->userName,
 "fullName" => $this->fullName,
 //uzupełnij pozostałe
 //...
];
 return $arr;
}

function save($plik){
 $tab = json_decode(file_get_contents($plik),true);
 array_push($tab,$this->toArray());
 file_put_contents($plik, json_encode($tab));
}
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



The screenshot shows a web browser window with the URL `localhost/phplab5/index.php`. At the top, there is a registration form with a text input field labeled "Email:" and two buttons: "Rejestruj" (Register) and "Anuluj" (Cancel). Below the form, a message says "Poprawne dane rejestracji:" followed by a list of registered users:

- beatap Beata Pańczyk b.panczyk@pollub.pl status: 1 2021-07-27
- admin Administrator 2018-09-10
- kubus Kubuś Puchatek 2019-10-30
- beatap Beata Pańczyk 2021-07-27

Rys. 5.4. Formularz i lista wszystkich zarejestrowanych użytkowników z pliku `users.json`

#### Zadanie 5.4. Obsługa formatu XML w PHP

Zadanie jest analogiczne do poprzedniego, tyle, że tym razem dane o użytkownikach będą zapisywane i odczytywane z pliku w formacie XML.

1. Do głównego folderu projektu dodaj nowy plik kategorii XML `users.xml` o zawartości jak na Listingu 5.8.
2. Do klasy `User` dodaj metodę `saveXML()`, która będzie zapisywała dane użytkownika w pliku XML (Listing 5.9).
3. Do klasy `User` dodaj kolejną metodę **statyczną** `getAllUsersFromXML()`, która będzie wyświetlała listę wszystkich zarejestrowanych użytkowników z pliku XML (Listing 5.10).
4. W skrypcie `index.php`, po wyświetleniu prawidłowych danych rejestracji – przetestuj zapis nowego użytkownika do pliku `users.xml` (`$user->saveXML()`). Na dole strony z formularzem wyświetl listę wszystkich zarejestrowanych w pliku `users.xml` użytkowników (`User::getAllUsersFromXML()`).

#### UWAGA!

W PHP obsługę formatu XML realizuje klasa **SimpleXML**. Obiekty tej klasy tworzą drzewo, którego struktura odpowiada strukturze kodu XML. Każdemu elementowi XML odpowiada jeden obiekt SimpleXML, natomiast atrybuty obiektu zwracane są w postaci tablicy asocjacyjnej. Konstruktor SimpleXMLElement oraz funkcje `simplexml_load_file()` i `simplexml_load_string()` tworzą drzewo obiektów SimpleXML na podstawie kodu XML podanego jako parametr.

#### Listing 5.8. Początkowa zawartość pliku `users.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
 <user>
 <userName>admin</userName>
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
<passwd>admin</passwd>
<fullName>Administrator</fullName>
<email>admin@gmail.com</email>
<date>2018-09-10</date>
<status>2</status>
</user>
<user>
 <userName>beatap</userName>
 <passwd>beatap</passwd>
 <fullName>Beata P.</fullName>
 <email>beatap@gmail.com</email>
 <date>2018-10-10</date>
 <status>1</status>
</user>
</users>
```

**Listing 5.9. Odczyt pliku XML i dodanie nowego elementu user**

```
//wczytujemy plik XML:
$xml = simplexml_load_file('users.xml');
//dodajemy nowy element user (jako child)
$xmlCopy=$xml->addChild("user");
//do elementu dodajemy jego właściwości o określonej nazwie i treści
$xmlCopy->addChild("userName", $this->userName);
//uzupełnij pozostałe właściwości
// ...
//zapisujemy zmodyfikowany XML do pliku:
$xml->asXML('users.xml');
```

**Listing 5.10. Odczyt wszystkich elementów user z pliku XML**

```
$allUsers = simplexml_load_file('users.xml');
echo "";
foreach ($allUsers as $user):
 $userName=$user->userName;
 $date=$user->date;
 echo "$userName, $date, ... ";
endforeach;
echo "";
```

Więcej informacji i przykładów współpracy PHP z XML możesz znaleźć na stronach:

[http://www.w3schools.com/php/php\\_xml\\_simplexml\\_read.asp](http://www.w3schools.com/php/php_xml_simplexml_read.asp)



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

## LABORATORIUM 6. PRACA Z DANYMI W PHP I MySQL. REALIZACJA KLASY DO OBSŁUGI BAZY DANYCH W OPARCIU O ROZSzerZENIE MySQLi LUB PDO. STWORZENIE PROSTEJ APLIKACJI TYPU CRUD.

### Cel laboratorium:

Celem zajęć jest poznanie narzędzia **phpMyAdmin** do zarządzania bazą danych MySQL oraz wykorzystanie interfejsów **mysqli** i **PDO** do wykonywania operacji na bazie danych.

### Zakres tematyczny zajęć:

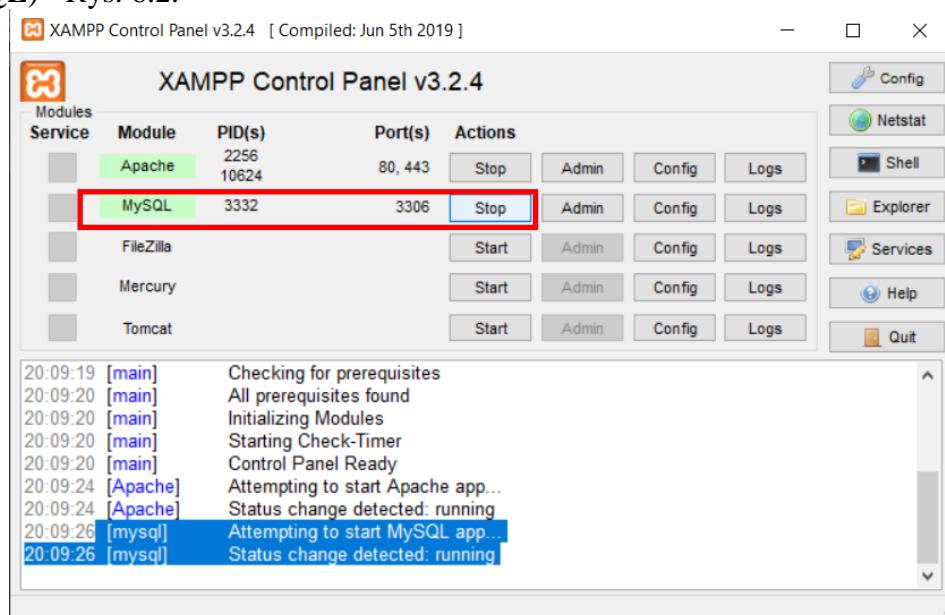
Poznanie narzędzia **phpMyAdmin**. Przygotowanie projektu aplikacji, która umożliwia składanie zamówień przechowywanych w bazie danych, ich przeglądanie i usuwanie. Opracowanie pomocniczej klasy do wykonywania operacji na dowolnej bazie danych. Do składania zamówienia zostanie wykorzystany formularz opracowany w ramach laboratorium 3, walidowany na laboratorium 4.

### Pytania kontrolne:

1. Co jest niezbędne do programowania aplikacji po stronie serwera?
2. Czym różni się programowanie aplikacji typu client-side od server-side?
3. Za pomocą jakiego znacznika osadza się skrypty PHP w kodzie dokumentu HTML?
4. Jakie są podstawowe typy danych i instrukcje w PHP?

### Zadanie 6.1. Tworzenie bazy danych w phpMyAdmin

1. Uruchom serwer **Apache** i serwer **MySQL** (okno XAMPP Control Panel) - Rys. 6.1. W przeglądarce wpisz adres *localhost* i z górnego menu strony głównej XAMPP wybierz opcję **phpMyAdmin** (narzędzie wspierające szybkie zarządzanie danymi na serwerze **MySQL**) - Rys. 6.2.



Rys. 6.1. Uruchomienie serwera MySQL za pomocą panelu kontrolnego w XAMPP



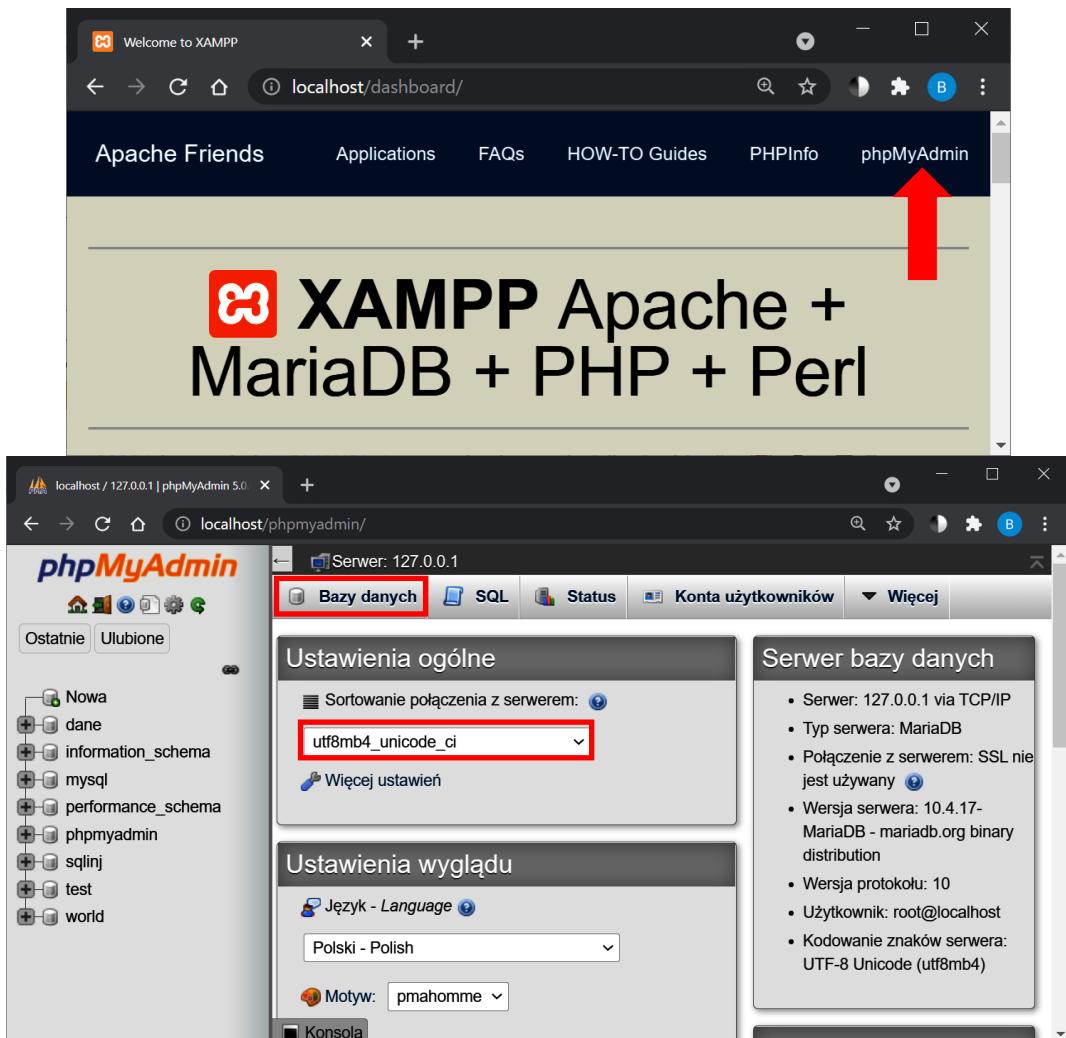
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

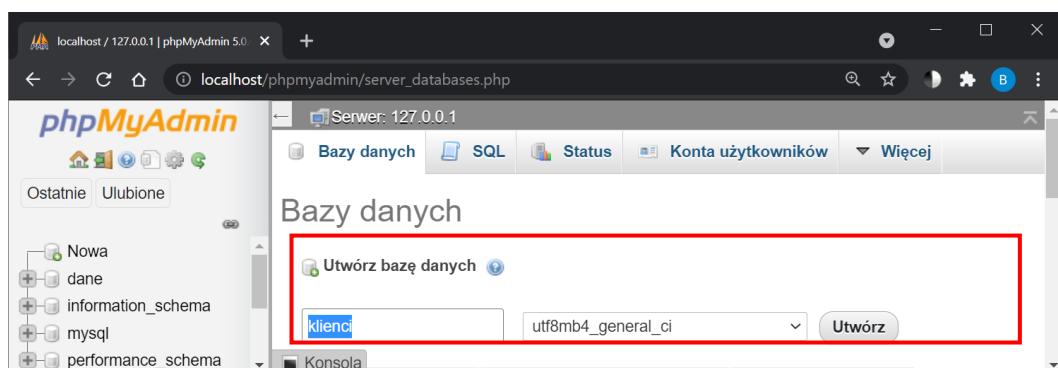
Unia Europejska  
Europejski Fundusz Społeczny





Rys. 6.2. Uruchomienie narzędzia PHPMyAdmin

2. Utwórz nową bazę danych o nazwie **klienci** (zwróć uwagę na metodę porównywania napisów **utf8mb4\_general\_ci**) (Rys. 6.3). Po utworzeniu powinna się ona pojawić na liście baz dostępnych na serwerze (pasek z lewej strony okna).



Rys. 6.3. Tworzenie nowej bazy danych **klienci**

3. Wskaż bazę **klienci** i dodaj do niej tabelę o tej samej nazwie **klienci** o 7 polach (wykorzystaj polecenie SQL z Listingu 6.1 i opcję SQL w phpMyAdmin): **Id** (**INT**,



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

**PRIMARY KEY, AUTO\_INCREMENT, UNSIGNED**, **Nazwisko** (VARCHAR(40)), **Wiek** (TINYINT, UNSIGNED), **Panstwo** (ENUM), **Email** (VARCHAR(40)), **Zamowienie** (SET), **Platnosc** (ENUM). Do definicji typów pól wykorzystano typy danych MySQL (Rys. 6.4) np.:

**VARCHAR** - pole tekstowe zmiennej długości,

**TINYINT** - liczby z zakresu 0-256 (jeśli **UNSIGNED**),

**ENUM** - stosuje się gdy w polu może się znaleźć jedna z podanych na liście wartości,  
**SET** - tak jak **ENUM**, ale może to być kilka wartości równocześnie.

Wartości dla **ENUM** podaje się jako: 'wartosc1', 'wartosc2', 'wartosc3' a wartości domyślne wpisujemy po prostu: **wartosc1**.

#### Listing 6.1. Polecenie SQL do tworzenia tabeli klienci

```
CREATE TABLE IF NOT EXISTS `klienci` (
 `Id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
 `Nazwisko` varchar(40) NOT NULL ,
 `Wiek` tinyint(3) UNSIGNED NOT NULL,
 `Panstwo` enum('Polska','Niemcy','Wielka Brytania','Czechy') NOT NULL
 DEFAULT 'Polska',
 `Email` varchar(50) NOT NULL,
 `Zamowienie` set('Java','PHP','CPP') NOT NULL DEFAULT 'PHP',
 `Platnosc` enum('Visa','Master Card','Przelew') NOT NULL DEFAULT
 'Visa',
 PRIMARY KEY (`Id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

| # | Nazwa             | Typ                                                   | Metoda porównywania napisów | Atrybuty | Null | Ustawienia domyślne | Komentarze | Dodatkowo      |
|---|-------------------|-------------------------------------------------------|-----------------------------|----------|------|---------------------|------------|----------------|
| 1 | <b>Id</b>         | int(10)                                               |                             | UNSIGNED | Nie  | Brak                |            | AUTO_INCREMENT |
| 2 | <b>Nazwisko</b>   | varchar(40)                                           | utf8_general_ci             |          | Nie  | Brak                |            |                |
| 3 | <b>Wiek</b>       | tinyint(3)                                            |                             | UNSIGNED | Nie  | Brak                |            |                |
| 4 | <b>Panstwo</b>    | enum('Polska','Niemcy','Wielka Brytania','Czechy...') |                             |          | Nie  | Polska              |            |                |
| 5 | <b>Email</b>      | varchar(50)                                           | utf8_general_ci             |          | Nie  | Brak                |            |                |
| 6 | <b>Zamowienie</b> | set('Java','PHP','CPP')                               | utf8_general_ci             |          | Nie  | PHP                 |            |                |
| 7 | <b>Platnosc</b>   | enum('Visa','Master Card','Przelew')                  | utf8_general_ci             |          | Nie  | Visa                |            |                |

Rys.6.4. Struktura tabeli klienci



4. Dodaj w phpMyAdmin kilka rekordów do stworzonej tabeli. Zwróć uwagę na składnię zapytania typu INSERT, które można obserwować po każdym wykonaniu polecenia z wykorzystaniem interfejsu graficznego w phpMyAdmin. Umiejętność sformułowania poprawnego zapytania SQL będzie niezbędna przy pracy z bazą danych z poziomu skryptu PHP.

### Zadanie 6.2. Praca z PHP i MySQL

Stwórz nowy projekt, np. **phplab6**. Do zadania wykorzystaj formularz i pomocnicze funkcje z laboratorium 4, tyle że tym razem wszystkie operacje zamiast na pliku wykonaj na bazie danych (przycisk *dodaj* - dodaje rekord z zamówieniem do tabeli *klienci*, przycisk *pokaz* - wyświetla wszystkie rekordy z tabeli *klienci* itd.). Do operacji na bazie danych skorzystaj z pomocniczej klasy (Listing 6.2). Definicję klasy zapisz w folderze **klasy** w pliku **Baza.php**. Na początek przetestuj skrypt (Listing 6.3), który korzystając z metod klasy **Baza** wyświetli wszystkie dane pobrane z tabeli *klienci*.

Metody *delete()* i *insert()* w klasie *Baza* wymagają uzupełnienia (patrz wykład PHP i MySQL).

Zauważ ponadto, że zapytanie SELECT można wykonać również z wykorzystaniem dopasowania do wyrażenia regularnego. Ma ono wtedy postać np.:

**SELECT kolumna1,kolumna2 FROM tabela WHERE kolumna2 REGEXP 'Java'**

Pamiętaj, żeby przed dodaniem danych do bazy sprawdzić ich poprawność, stosując odpowiednie filtry.

**Listing 6.2. Klasa Baza.php**

```
<?php
class Baza {
 private $mysqli; //uchwyt do BD
 public function __construct($serwer, $user, $pass, $baza) {
 $this->mysqli = new mysqli($serwer, $user, $pass, $baza);
 /* sprawdz połączenie */
 if ($this->mysqli->connect_errno) {
 printf("Nie udało się połączenie z serwerem: %s\n",
 $mysqli->connect_error);
 exit();
 }
 /* zmien kodowanie na utf8 */
 if ($this->mysqli->set_charset("utf8")) {
 //udało się zmienić kodowanie
 }
 } //koniec funkcji konstruktora
 function __destruct() {
 $this->mysqli->close();
 }
 public function select($sql, $pola) {
 //parametr $sql - łańcuch zapytania select
 //parametr $pola - tablica z nazwami pol w bazie
 //Wynik funkcji - kod HTML tabeli z rekordami (String)
 $tresc = "";
 }
}
```



```
if ($result = $this->mysqli->query($sql)) {
 $ilepol = count($pola); //ile pól
 $ile = $result->num_rows; //ile wierszy
 // pętla po wyniku zapytania $results
 $tresc.= "<table><tbody>";
 while ($row = $result->fetch_object()) {
 $tresc.= "<tr>";
 for ($i = 0; $i < $ilepol; $i++) {
 $p = $pola[$i];
 $tresc.= "<td>" . $row->$p . "</td>";
 }
 $tresc.= "</tr>";
 }
 $tresc.= "</tbody></table>";
 $result->close(); /* zwolnij pamięć */
}
return $tresc;
}
public function insert($sql) {
 // uzupełnij zapytanie - i zwróć true lub false
}
public function delete($sql) {
 // uzupełnij zapytanie - i zwróć true lub false
}
public function insert($sql) {
 if($this->mysqli->query($sql)) return true; else return false;
}
public function getMysqli() {
 return $this->mysqli;
}
} //koniec klasy Baza
```

Listing 6.3. Schemat skryptu index.php

```
<!-- Pokaż formularz zamówienia -->
<?php
 include_once("funkcje.php");
 include_once "klasy/Baza.php";
 //tworzymy uchwyt do bazy danych:
 $bd = new Baza("localhost", "root", "", "klienci");
 if (filter_input(INPUT_POST, "submit")) {
 $akcja = filter_input(INPUT_POST, "submit");
 switch ($akcja) {
 case "Dodaj" : dodajdoBD($bd); break;
 case "Pokaż" : echo $bd->select("select Nazwisko,Zamowienie
 from klienci", ["Nazwisko","Zamowienie"]); break;
 }
 ?>
```



Dodatkową funkcję, którą należy dodać do pliku *funkcje.php* przedstawia Listing 6.3.

**Listing 6.3. Dodatkowa funkcja dodajdoBD**

```
function dodajdoBD($bd){
 //pobierz dane z formularza, dokonaj ich walidacji
 //jeśli dane są poprawne - sformułuj zapytanie $sql typu insert np.:
 // INSERT INTO klienci VALUES (NULL, 'Babacka', '22', 'Niemcy',
 // 'bbbp@ollub.pl', 'Java,CPP', 'Przelew');
 //i wywołaj metodę:
 //{$bd->insert($sql);
}
}
```

Przykładowy efekt działania aplikacji przedstawia Rys. 6.5.

 string(10) "Kalinowska" ["wiek"]=> int(51) ["kraj"]=> string(2) "pl" ["email"]=> string(17) "kalinka@gmail.com" ["jezyki"]=> array(2) { [0]=> string(3) "PHP" [1]=> string(10) "JavaScript" } ["zaplata"]=> string(4) "visa" }. Below this is the message 'Dodano do bazy'."/>

**Formularz zamówienia**

Nazwisko:

Wiek:

Państwo:

Adres e-mail:

**Zamawiam tutorial z języka:**

C  CPP  Java  C#  HTML  CSS  XML  PHP  JavaScript

**Sposób zapłaty:**

eurocard  visa  przelew bankowy

```
array(6) { ["nazw"]=> string(10) "Kalinowska" ["wiek"]=> int(51) ["kraj"]=> string(2) "pl" ["email"]=> string(17) "kalinka@gmail.com" ["jezyki"]=> array(2) { [0]=> string(3) "PHP" [1]=> string(10) "JavaScript" } ["zaplata"]=> string(4) "visa" }
Dodano do bazy
```

Rys.6.5. Przykładowy widok aplikacji współpracującej z bazą danych MySQL

Alternatywnie operację dodawania danych do bazy można zrealizować za pomocą metod: *\$bd->prepare(...)* oraz *bindParam(...)*. Sprawdź ich zastosowanie.

Przykład pokazujący wykorzystanie tych funkcji przedstawia Listing 6.4.

**Listing 6.4. Instrukcja INSERT z zastosowaniem funkcji prepare i bindParam**

```
<?php
$stmt = $bd->prepare("INSERT INTO USERS (login, pass) VALUES (?, ?)");
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



```
$stmt->bindParam(1, $login);
$stmt->bindParam(2, $pass);
// dodanie jednego wiersza:
$login = 'ola';
$pass = 'ola123';
$stmt->execute();

// dodanie kolejnego wiersza z innymi wartościami:
$login = 'jasio';
$pass = 'jasio123';
$stmt->execute();
?>
```

### Zadanie 6.3. Praca z interfejsem PDO

Zrealizuj zadanie 6.2. tak, aby operacje na danych były realizowane za pomocą interfejsu PDO. Możesz skorzystać z przykładu wykładowego i zdefiniować klasę **BazaPDO**, ale tym razem wykonując operacje na danych za pomocą metod interfejsu PDO. Efekt działania aplikacji powinien być taki sam jak w przypadku zadania 6.2.

Fragment klasy BazaPDO przedstawia Listing 6.5.

#### Listing 6.5. Schemat klasy BazaPDO

```
private $dbh; //uchwyt do BD
public function __construct($serwer, $user, $pass) {
 try { $this->dbh = new PDO($serwer, $user, $pass,
 [PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"]); }
 catch (PDOException $e) {
 print "Error!: " . $e->getMessage() . "
"; die();
 }
} //koniec funkcji konstruktora
function __destruct() {
 $this->dbh=null;
}
public function select($sql) {
 //parametr $sql – łańcuch zapytania select
 foreach ($this->dbh->query($sql) as $row) { print_r($row); }
}
} //koniec klasy BazaPDO
?>
```



## LABORATORIUM 7. IMPLEMENTACJA MECHANIZMU SESJI W APLIKACJACH PHP. ZASTOSOWANIE CIASTECZEK.

### Cel laboratorium:

Celem zajęć jest poznanie i wykorzystanie w praktyce mechanizmu emulacji sesji HTTP.

### Zakres tematyczny zajęć:

Budowa prostego skryptu PHP z zastosowaniem podstawowych mechanizmów do obsługi sesji HTTP. Do realizacji zadań wykorzystana będzie klasa **User** oraz formularz rejestracyjny z laboratorium 5 oraz klasa **Baza** z laboratorium 6.

### Pytania kontrolne:

1. Na czym polega emulacja sesji HTTP i do czego służy?
2. Gdzie może być przechowywany identyfikator sesji?
3. Jakie typy wartości można przechowywać w tablicy `$_SESSION`?

### Zadanie 7.1. Utworzenie tabeli users w bazie klienci

Skorzystaj z projektu utworzonego na laboratorium 5. W *PhpMyAdmin* utwórz tabelę **users** (we wcześniej stworzonej bazie danych **klienci**, wykorzystując skrypt z Listingu 7.1).

#### Listing 7.1. Polecenie SQL tworzenia tabeli users

```
CREATE TABLE IF NOT EXISTS `users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`userName` varchar(100) NOT NULL,
`fullName` varchar(255) NOT NULL,
`email` varchar(100) NOT NULL,
`passwd` varchar(255) NOT NULL,
`status` int(1) NOT NULL,
`date` datetime NOT NULL,
PRIMARY KEY (`id`),
UNIQUE KEY `userName`(`userName`),
(`userName`, `email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
AUTO_INCREMENT=1 ;
```

### Zadanie 7.2. Rozbudowa klasy User

Zmodyfikuj aplikację utworzoną na laboratorium 5 – zamiast rejestrować użytkowników z zapisem ich danych do pliku, dane rejestracyjne będziemy przechowywać w bazie danych **klienci** w tabeli **users**. Korzystając z klas **User**, **RegistrationForm** i skryptu z formularzem rejestracyjnym, do tabeli **users** dodamy kilku użytkowników.

W tym celu:

- a) do klasy **User** dodaj dwie dodatkowe metody z parametrem `$db` (uchwyt do bazy danych): `saveDB($db)` (zapis do tabeli **users**) i `getAllUsersFromDB($db)` (metoda statyczna - pobiera i wyświetla dane wszystkich użytkowników z tabeli **users**). Do operacji na bazie danych skorzystaj z klasy **Baza** z Lab. 6. Sama klasa **Baza** nie



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

wymaga już żadnych modyfikacji, należy tylko podać właściwe parametry połączenia tworząc nowy obiekt klasy *Baza* oraz operować właściwymi parametrami metod *select()* czy *insert()* klasy *Baza*. Zauważ ponadto, że parametrem metod *insert()*, czy *delete()* w tej klasie jest tylko odpowiednio sformułowany łańcuch **zapytania SQL** odpowiednio typu **INSERT** lub **DELETE**. Klasa *Baza* jest w pełni uniwersalna i nie zależy od konkretnej nazwy bazy danych czy tabeli.

- b) schemat skryptu *index.php* z formularzem rejestracyjnym pozostaje analogiczny jak w zadaniu z laboratorium 5, tylko **pamiętaj o dołączeniu klasy Baza** oraz zmień metodę zapisu nowego użytkownika i odczytu danych – zapis zrealizuj teraz do bazy danych (metodą *saveDB()*) i odczyt z bazy danych (metodą *getAllUsersFromDB()*).

Sprawdź działanie skryptu *index.php*.

### Zadanie 7.3. Mechanizm działania sesji

Zmienne sesyjne w języku PHP są umieszczane w specjalnej tablicy asocjacyjnej **\$\_SESSION** i są dostępne dla wszystkich stron należących do danej domeny. Możemy z nich korzystać np. w celu przekazywania informacji pomiędzy stronami czy w celu uwierzytelnienia użytkownika.

- a) Przetestuj mechanizm działania sesji wykonując następujący skrypt umieszczony na stronie *test1.php*:

```
session_start();
$_SESSION['username'] = 'kubus';
$_SESSION['fullname'] = 'Kubus Puchatek';
$_SESSION['email'] = 'kubus@stumilowylas.pl';
$_SESSION['status'] = 'ADMIN';
```

- b) Następnie na stronie *test1.php* (Rys.7.1):

- wyświetl **id** sesji,
- wyświetl **wszystkie** zmienne sesji (skorzystaj z pętli *foreach* i tablicy **\$\_SESSION**),
- wyświetl **ciasteczka** skojarzone z domeną *localhost* (zastosuj pętlę *foreach* dla tablicy **\$\_COOKIE**),
- umieść link do kolejnej strony *test2.php*.

- c) Korzystając z narzędzi deweloperskich dostępnych w przeglądarce - przejrzyj, jakie ciasteczka (pasek narzędzi deweloperskich i zakładka *Application*) zostały utworzone po uruchomieniu skryptu *test1.php* (Rys. 7.1).

- d) Przejrzyj zawartość folderu *xampp/tmp* (w przypadku XAMPP lub odpowiednio innego) i znajdź plik przechowujący zmienne utworzonej sesji (nazwa pliku zawiera id sesji, np. w niniejszym przykładzie nazwa pliku to: *sess\_50he1lkqhhc34onugi1mg9l890*). Przejrzyj jego zawartość.

- e) Na stronie *test2.php* (Rys. 7.2):

- wyświetl **id** sesji i wszystkie zmienne sesji,
- wykonaj usunięcie sesji (funkcja *session\_destroy()*),
- umieść link powrotny do strony *test1.php*,
- sprawdź czy **id** sesji jest taki sam jak na stronie *test1.php* i czy istnieje ciasteczko sesyjne z tym samym **id** (dostępne ze strony test2).



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

- f) Skorzystaj z linku powrotnego do strony **test1.php** i sprawdź, czy został zmieniony **id** sesji po wykonaniu funkcji **session\_destroy()** na stronie test2. Co należy zrobić, aby „zabić” sesję (sprawdź w materiałach wykładowych)?

The screenshot shows a browser window with the URL `localhost/phplab5/test1.php`. The page content displays session variables:

```
Id sesji:50he1lkqhhc34onugi1mg9l89o
username: kubus
username: Kubus Puchatek
email: kubus@stumilowylas.pl
status: ADMIN
```

Below the session variables, there is a "Ciasteczka:" section with the PHPSESSID value:

**Ciasteczka:**  
**PHPSESSID=50he1lkqhhc34onugi1mg9l89o**

At the bottom, there is a link: [Przejdz do strony 2](#).

Below the browser window, the developer tools Network tab is open, showing the Cookies section. A cookie named `PHPSESSID` is listed with the value `50he1lkqhhc34onugi1mg9l89o`.

| Name      | Value                      | D...  | Path | Ex... | Size | Htt... | Se... | Sa... | Pri... |
|-----------|----------------------------|-------|------|-------|------|--------|-------|-------|--------|
| PHPSESSID | 50he1lkqhhc34onugi1mg9l89o | lo... | /    | Se... | 35   |        |       |       | Me...  |

Rys. 7.1. Strona test1.php

The screenshot shows a browser window with the URL `localhost/phplab5/test2.php`. The page content displays session variables:

```
Id sesji:50he1lkqhhc34onugi1mg9l89o
username: kubus
username: Kubus Puchatek
email: kubus@stumilowylas.pl
status: ADMIN
```

Below the session variables, there is a "Ciasteczka:" section with the PHPSESSID value:

**Ciasteczka:**  
**PHPSESSID=50he1lkqhhc34onugi1mg9l89o**

At the bottom, there is a link: [Przejdz do strony 1](#).

Below the browser window, the developer tools Network tab is open, showing the Cookies section. A cookie named `PHPSESSID` is listed with the value `50he1lkqhhc34onugi1mg9l89o`.

| Name      | Value                      | D...  | Path | Ex... | Size | Htt... | Se... | Sa... | Pri... |
|-----------|----------------------------|-------|------|-------|------|--------|-------|-------|--------|
| PHPSESSID | 50he1lkqhhc34onugi1mg9l89o | lo... | /    | Se... | 35   |        |       |       | Me...  |

Rys. 7.2. Strona test2.php



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



#### **Zadanie 7.4. Przechowywanie obiektu w zmiennej sesji**

Zmodyfikuj działanie skryptów ze stron ***test1*** i ***test2***. Zamiast tworzyć 4 zmienne sesji, jak w punkcie 7.3a) - utwórz obiekt ***user*** klasy ***User*** i zapamiętaj go w zmiennej sesji pod kluczem "***user***" (zobacz przykład z wykładu).

Wyświetl właściwości **obiektu** ***user*** zapamiętanego w sesji na stronie ***test2.php***. Pamiętaj o konieczności skorzystania z funkcji ***serialize*** i ***unserialize***. Efekt działania powinien być analogiczny do tego z rys. 7.1 i 7.2.



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny



## LABORATORIUM 8. OBSŁUGA REJESTRACJI I LOGOWANIA UŻYTKOWNIKA

### Cel laboratorium:

Celem zajęć jest przygotowanie skryptów umożliwiających uwierzytelnienie użytkownika na podstawie jego danych przechowywanych w bazie danych za pomocą loginu i hasła.

### Zakres tematyczny zajęć:

Przygotowanie skryptów do obsługi rejestracji i logowania użytkownika z zastosowaniem klas i bazy danych przygotowanych na poprzednich zajęciach. Uwierzytelnienie użytkownika w kontroli sesji.

### Pytania kontrolne:

1. Na czym polega uwierzytelnianie użytkownika?
2. W jaki sposób jest przechowywane hasło w bazie danych?
3. W jaki sposób korzysta się z mechanizmu sesji http w celu uwierzytelnienia użytkownika?

### Zadanie 8.1. Uwierzytelnienie użytkownika w kontroli sesji

- a) W *PhpMyAdmin* w poprzednio tworzonej bazie *klienci* utwórz 2 tabele, korzystając ze skryptu (jeśli tabela *users* jest już utworzona na **Lab7** - utwórz tylko tabelę *logged\_in\_users*):

```
--
-- Struktura tabeli dla `users`
--
CREATE TABLE IF NOT EXISTS `users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`userName` varchar(100) NOT NULL,
`fullName` varchar(255) NOT NULL,
`email` varchar(100) NOT NULL,
`passwd` varchar(255) NOT NULL,
`status` int(1)
PRIMARY KEY (`id`),
UNIQUE KEY `userName`
(`userName`, `email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
AUTO_INCREMENT=1 ;

--
-- Struktura tabeli dla `logged_in_users`
--
CREATE TABLE IF NOT EXISTS `logged_in_users` (
`sessionId` varchar(100) NOT NULL,
`userId` int(11) NOT NULL,
`lastUpdate` datetime NOT NULL,
PRIMARY KEY (`sessionId`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

- b) Korzystając z przykładów wykładowych oraz klas opracowanych na poprzednich zajęciach (**User** i **Baza**) utwórz mechanizm logowania dla użytkowników, wykonując następujące kroki:
- Do klasy **Baza** dodaj kolejną metodę pomocniczą **selectUser(\$login, \$passwd, \$tabela)**, która zwraca **\$id** użytkownika lub -1, jeśli taki nie został znaleziony w zadanej tabeli (Listing 8.1). Sprawdź także, czy w klasie **Baza** znajduje się uzupełniona metoda **delete()**.
  - Utwórz pomocniczą klasę do zarządzania logowaniem **UserManager**, która zawiera metodę **loginForm()** do wyświetlenia formularza logowania oraz metody **login(\$db)**, **logout(\$db)** i **getLoggedInUser(\$db, \$id)** (Listing 8.2):
    - metoda **login(\$db)** najpierw filtry dane przesłane z formularza logowania, a następnie sprawdza, czy dany użytkownik znajduje się w tabeli **users** w bazie, do której uchwyt jest podany za pomocą parametru **\$db**. Jeżeli dane logowania są poprawne - metoda rozpoczyna sesję, czyści wszystkie wpisy historyczne logowania w tabeli **logged\_in\_users** dla danego użytkownika oraz zapisuje aktualne dane logowania do tabeli **logged\_in\_users**. Metoda zwraca w wyniku **id** zalogowanego użytkownika lub -1.
    - metoda **logout(\$db)** usuwa z tabeli **logged\_in\_users** rekord z **id** bieżącej sesji oraz likwiduje sesję.
    - metoda **getLoggedInUser(\$db, \$sessionId)** sprawdza, czy użytkownik się zalogował - jako parametr przyjmuje **id** sesji i sprawdza czy dla tego id istnieje wpis w tabeli **logged\_in\_users**. Zwraca **id zalogowanego użytkownika** lub -1.
  - c) Przetestuj stworzony mechanizm uwierzytelniania w skrypcie **loginProcess.php** (Listing 8.3). Obserwuj na bieżąco stan tabeli **logged\_in\_users**.

**Listing 8.1. Nowa metoda selectUser w klasie Baza**

```
public function selectUser($login, $passwd, $tabela) {
 //parametry $login, $passwd , $tabela - nazwa tabeli z użytkownikami
 //wynik - id użytkownika lub -1 jeśli dane logowania nie są poprawne
 $id = -1;
 $sql = "SELECT * FROM $tabela WHERE userName='$login'";
 if ($result = $this->mysqli->query($sql)) {
 $ile = $result->num_rows;
 if ($ile == 1) {
 $row = $result->fetch_object() //pobierz rekord z użytkownikiem
 $hash = $row->passwd; //pobierz zahaszowane hasło użytkownika
 //sprawdź czy pobrane hasło pasuje do tego z tabeli bazy danych:
 if (password_verify($passwd, $hash))
 $id = $row->id; //jeśli hasła się zgadzają - pobierz id
użytkownika
 }
 }
 return $id; //id zalogowanego użytkownika(>0) lub -1
}
```



**Listing 8.2. Schemat klasy UserManager**

```
class UserManager {
 function loginForm() {
 ?
 <h3>Formularz logowania</h3><p>
 <form action="processLogin.php" method="post">
 ...
 <input type="submit" value="Zaloguj" name="zaloguj" />
 </form></p> <?php
 }
 function login($db) {
 //funkcja sprawdza poprawność logowania
 //wynik - id użytkownika zalogowanego lub -1
 $args = [
 'login' => FILTER_SANITIZE_MAGIC_QUOTES,
 'passwd' => FILTER_SANITIZE_MAGIC_QUOTES
];
 //przefiltruj dane z GET (lub z POST) zgodnie z ustawionymi w $args
 filtrami:
 $dane = filter_input_array(INPUT_POST, $args);
 //sprawdź czy użytkownik o loginie istnieje w tabeli users
 //i czy podane hasło jest poprawne
 $login = $dane["login"];
 $passwd = $dane["passwd"];
 $userId = $db->selectUser($login, $passwd, "users");
 if ($userId >= 0) { //Poprawne dane
 //rozpocznij sesję zalogowanego użytkownika
 //usuń wszystkie wpisy historyczne dla użytkownika o $userId
 //ustaw datę - format("Y-m-d H:i:s");
 //pobierz id sesji i dodaj wpis do tabeli logged_in_users
 }
 return $userId;
 }
 function logout($db) {
 //pobierz id bieżącej sesji (pamiętaj o session_start())
 //usuń sesję (łącznie z ciasteczką sesyjnym)
 //usuń wpis z id bieżącej sesji z tabeli logged_in_users
 }
 function getLoggedInUser($db, $sessionId) {
 //wynik $userId - znaleziono wpis z id sesji w tabeli
 logged_in_users
 //wynik -1 - nie ma wpisu dla tego id sesji w tabeli
 logged_in_users
 }
}
```

**Listing 8.3. Schemat skryptu processLogin.php**

```
<?php
 include_once 'klasy/Baza.php';
 include_once 'klasy/User.php';
 include_once 'klasy/UserManager.php';
 $db = new Baza("localhost", "root", "", "klienci");
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

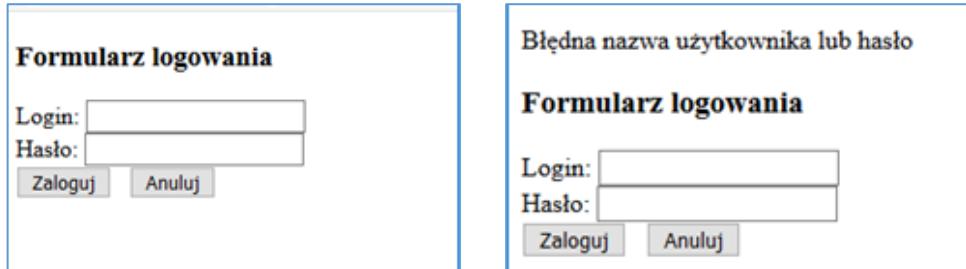


```

$um = new UserManager();
//parametr z GET - akcja = wyloguj
if (filter_input(INPUT_GET, "akcja")=="wyloguj") {
 $um->logout($db);
}
//kliknięto przycisk submit z name = zaloguj
if (filter_input(INPUT_POST, "zaloguj")) {
 $userId=$um->login($db); //sprawdź parametry logowania
 if ($userId > 0) {
 echo "<p>Poprawne logowanie.
";
 echo "Zalogowany użytkownik o id=$userId
";
 //pokaż link wyloguj lub przekieruj
 // użytkownika na inną stronę dla zalogowanych
 echo "
 Wyloguj </p>";
 } else {
 echo "<p>Błędna nazwa użytkownika lub hasło</p>";
 $um->loginForm(); //Pokaż formularz logowania
 }
} else {
 //pierwsze uruchomienie skryptu processLogin
 $um->loginForm();
}
?>

```

Przykładowe efekty działania uwierzytelnienia użytkownika przedstawiają rysunki 8.1 i 8.2.



Rys. 8.1. Przykładowy efekt działania skryptu processLogin.php

| sessionId                  | userId | lastUpdate          |
|----------------------------|--------|---------------------|
| e12mnicsruor7mstu3k5eehuf4 | 13     | 2018-11-24 11:01:52 |
| jeq8bm0qblvuqnqfsf4eo09ecp | 15     | 2018-11-24 12:01:50 |
| uuigfjlrhid6ldl3q8a5j1b0rj | 14     | 2018-11-24 11:36:47 |

Rys. 8.2. Poprawne dane logowania i rekordy w tabeli logged\_in\_users

### Zadanie 8.2.

- Zmodyfikuj działanie skryptu **processLogin.php**, tak aby po poprawnym zalogowaniu użytkownik został przekierowany na stronę dla zalogowanych użytkowników (np. **testLogin.php**). W tym celu należy zastosować funkcję: **header("location:testLogin.php");**



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



- b) Na stronie ***testLogin.php*** (pamiętaj o pobraniu plików z definicją odpowiednich klas) **na poczatku** należy sprawdzić, czy rzeczywiście w tabeli ***logged\_in\_users*** istnieje wpis z ***id*** bieżącej sesji.

Jeśli taki wpis istnieje:

- dodaj link do wylogowania (***processLogin.php?akcja=wyloguj***);
- wyświetl na stronie dane użytkownika z tabeli ***users*** o identyfikatorze pobranym z tabeli ***logged\_in\_users*** dla bieżącej sesji (Rys. 8.3).

Jeśli taki wpis nie istnieje – przekieruj użytkownika ponownie do strony z formularzem logowania (***processLogin.php***).

- c) Po wylogowaniu (lub bez wcześniejszego logowania użytkownika) spróbuj otworzyć w przeglądarce bezpośrednio stronę ***testLogin.php***. Jeśli nastąpiło przekierowanie do formularza logowania – strona jest zabezpieczona przed dostępem dla nieuprawnionych użytkowników.

The screenshot shows a web page with a blue header bar. Below it, the text "Wyloguj" (Logout) is displayed in blue. Underneath, the text "Dane zalogowanego użytkownika:" (Data of the logged-in user:) is shown in bold black font. Below this, the email address "13 beata bp@gmail.com" is listed. At the bottom, there is a link "I inne informacje dla zalogowanego użytkownika ..." (And other information for the logged-in user ...).

Rys. 8.3. Widok strony *testLogin.php* dla zalogowanego użytkownika



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## LABORATORIUM 9. IMPLEMENTACJA MECHANIZMÓW ZABEZPIECZEŃ PRZED POPULARNYMI ATAKAMI NA APLIKACJE INTERNETOWE

### Cel laboratorium:

Celem zajęć jest zapoznanie studentów z podstawowymi atakami realizowanymi na aplikacje internetowe oraz poznanie metod zabezpieczających przed nimi.

### Zakres tematyczny zajęć:

Przygotowanie skryptów demonstrujących możliwości przeprowadzenia typowych ataków typu **Cross Site Scripting**, **HTML Injection**, **Directory Traversal**, **File Inclusion**, **SQL Injection**, **Cross-Site Request Forgery**, **Session Hijacking**. Implementacja metod zabezpieczających skrypty przed tymi atakami.

### Pytania kontrolne:

1. Wymień i scharakteryzuj podstawowe ataki sieciowe.
2. Jakie znasz metody zabezpieczeń skryptów przed atakami na aplikacje internetowe?

Do realizacji poniższych ćwiczeń stwórz nowy projekt o nazwie np. **PHPBezpieczenstwo**.

### Zadanie 9.1. Niebezpieczne rozszerzenie (.inc, .txt) oraz przechowywanie kluczowych plików.

Unikaj nadawania plikom dołączanym do PHP rozszerzenia innego niż **.php**, ponieważ jest możliwe bezpośrednie wyświetlenie kodu źródłowego takiego pliku.

Jest to szczególnie niebezpiecznie, gdy w pliku znajdują się hasła np. do bazy danych.

Utwórz nowy plik **haslo.txt** ze skryptem:

```
<?php
 $host='localhost';
 $user='root';
 $pass='toor';
 $dbname='testowa';
 echo 'Test rozszerzenia .txt';
?>
```

Następnie spróbuj wykonać skrypt wpisując w pasku adresowym przeglądarki ścieżkę i nazwę tego pliku:

localhost/PHPBezpieczenstwo/hasla.txt

Co zostało wyświetlone w oknie przeglądarki?

Utwórz nowy plik **polaczenie.php**:

```
<?php
 include 'haslo.txt';
?>
```

Uruchom skrypt **polaczenie.php**.

W efekcie wykonania tego skryptu wykonała się instrukcja *echo* ze skryptu **haslo.txt**. Jednak nadal można podejrzeć kod skryptu z hasłem.

Aby to wyeliminować zmień rozszerzenie pliku **haslo.txt** na **haslo.php** oraz popraw rozszerzenie w pliku **polaczenie.php**. Uruchom ponownie oba skrypty.

Jeżeli serwer jest prawidłowo skonfigurowany, to nie wyświetli się już kod pliku, lecz tylko wykona się instrukcja *echo*.

Jednak nadal istnieje możliwość wyświetlenia zawartości pliku.

Żeby zminimalizować to niebezpieczeństwo, skopiuj plik **hasla.php** do katalogu znajdującego się powyżej katalogu **htdocs** (zakładamy, że pliki naszej aplikacji umieszczone są na serwerze w katalogu **htdocs**). Popraw także linię w pliku **polaczenie.php**:

```
include 'haslo.php';
```

na:

```
include '../haslo.php';
```

Wykonaj skrypt **polaczenie.php**, a następnie wpisz do paska adresowego:

```
http://localhost/PHPBezpieczenstwo/.../haslo.php
```

Serwer powinien zwrócić błąd.

Problemem jest to, że nie na wszystkich serwerach możemy wgrać plik do katalogu znajdującego się powyżej, co jest wynikiem konfiguracji serwera, który nie pozwala użytkownikowi na swobodne eksplorowanie katalogów.

### Zadanie 9.2. Atak typu Cross-Site Scripting oraz HTML Injection.

Utwórz następujący plik o nazwie *css.php*:

```
<form action="css.php" method="post">
 <textarea name="tekst"></textarea>

 <input type="submit" name="wyslij" value="Wyślij" />
</form>
<div>
<?php
 if (filter_input(INPUT_POST, 'wyslij'))
 echo $_POST['tekst'];
?
</div>
```

Uruchom skrypt i wpisz do pola *tekst* dowolny kod HTML, np. odnośnik do strony:

```
cs.pollub.pl
```

a następnie wyslij formularz.

Co się stało po wysłaniu formularza? Przejrzyj źródło strony (Ctrl+U).

Następnie zmodyfikuj skrypt tak, aby znaczniki HTML były zamieniane na postać niewykonywaną (tzn. znaki < oraz > mają być zamienione na encje &lt; oraz &gt;, a znak & w encjach zostanie zamieniony na &amp;).

Zmodyfikuj w tym celu linię z poleceniem *echo*:

```
echo htmlspecialchars($_POST['tekst']);
```

Ponownie przetestuj skrypt. Skrypt powinien teraz wyświetlić wprowadzony kod HTML, a nie wykonać go. Ponownie przejrzyj źródło strony.

Teraz zamiast `htmlspecialchars()` przetestuj funkcję `strip_tags()`, która usuwa wszystkie znaczniki HTML z podanego na atak ciągu. Czy widać na czym polega różnica w działaniu obu funkcji `htmlspecialchars()` i `strip_tags()`?

Zamiast tych funkcji skorzystaj z odpowiadających im filtrów:  
FILTER\_SANITIZE\_FULL\_SPECIAL\_CHARS i FILTER\_SANITIZE\_STRING  
oraz z funkcji `filter_input`.

### Zadanie 9.3. Atak typu Directory Traversal

Utwórz skrypt `DirTrav.php`:

```
<?php
function download($patch = '.')
{
 $katalog = @dir($patch) or die ('Brak dostępu do katalogu.');
 while ($plik_kat = $katalog->read())
 if(is_file($patch.'/'.$plik_kat))
 echo ''.$plik_kat.'
';
 elseif (is_dir($patch.'/'.$plik_kat))
 {
 echo ''.$plik_kat.'

';
 }
 $katalog->close();
}
if (!isset($_GET['patch']))
 download();
else
 download($_GET['patch']);
?>
```

Uruchom skrypt. Wyświetli się lista plików i katalogów z poziomu katalogu, w którym znajduje się skrypt. Spróbuj przejść do folderu nadzawanego klikając "..".

Czy możemy dowolnie poruszać się po strukturze katalogów serwera Apache?

Zabezpiecz skrypt zmieniając linię:

```
 download($_GET['patch']);
```

na

```
 download(str_replace('..','',$_GET['patch']));
```

Przetestuj teraz próbę cofnięcia się do katalogu nadzawanego. Powinna zakończyć się ona wyświetleniem zawartości katalogu zawierającego skrypt.

### Zadanie 9.4. Atak typu File Inclusion.

Jest to jeden z najpoważniejszych ataków, ponieważ atakujący może uruchomić dowolny skrypt PHP. Może na przykład uzyskać informacje na temat naszej aplikacji, podejrzeć kod skryptów PHP czy hasła do bazy danych (pomimo zabezpieczeń wprowadzonych w zadaniu 9.1). W tym ćwiczeniu złośliwy skrypt będzie jedynie wyświetlał tekst, ale w rzeczywistym ataku może to być skrypt, którego zadaniem będzie uzyskanie informacji o aplikacji.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

Utwórz plik *inclusion.php*:

```
<?php
 if (isset($_GET['plik']))
 include($_GET['plik'].'.txt');
 else
 echo 'Nie podano pliku!';
?>
```

Utwórz także plik tekstowy *skrypt.txt* i umieść go poza katalogiem WWW, najlepiej w *htdocs*:

```
<?php
 echo 'Strona nie jest zabezpieczona';
```

Wpisz do paska adresowego następujący adres:

localhost/PhpBezpieczenstwo/inclusion.php?plik=http://localhost/skrypt

Wykonał się kod PHP z pliku *skrypt.txt* lub pojawił się błąd spowodowany konfiguracją PHP. Żeby zobaczyć efekt ataku, należy w pliku *php.ini* zmodyfikować:

allow\_url\_include = Off

na:

allow\_url\_include = On

Po tej modyfikacji zrestartuj serwer Apache i ponownie przetestuj skrypt. Tym razem próba ataku się powiedzie.

Następnie zabezpiecz skrypt modyfikując:

```
if (isset($_GET['plik']))
 include($_GET['plik'].'.txt');
```

na:

```
if (isset($_GET['plik']))
{
 $plik=str_replace('\\','/',$_GET['plik']);
 $plik=str_replace('.','/',$plik);
 $filearr=explode('/',$plik);
 $plik=$filearr[count($filearr)-1];
 include($plik.'.txt');
}
```

Uruchom zmodyfikowany skrypt i spróbuj ponownie wykonać atak – tym razem pojawi się błąd o braku możliwości dołączenia pliku (jeśli *skrypt.txt* znajduje się w innym katalogu niż *inclusion.php*).

Po zakończeniu zadania przywróć ustawienia w *php.ini* na:

allow\_url\_include = Off.

### Zadanie 9.5. Atak typu SQL Injection.

Jest to jeden z najpoważniejszych ataków na aplikacje internetowe. Atakujący może np. wykasować rekordy lub tabele, zmienić i wstawiać nowe rekordy, co może prowadzić do przejęcia kontroli nad stroną.

Jeśli atakujący zdoła dodać użytkownika do bazy z prawami administratora lub będąc zwykłym użytkownikiem zmieni uprawnienia na administratora, to będzie mógł zarządzać naszym systemem.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

Przed wykonaniem ćwiczenia sprawdź, czy opcja ***magic\_quotes\_gpc*** w pliku ***php.ini*** jest ustawiona na ***Off***. Jeżeli jest ***On***, to wyłącz ją na czas tego ćwiczenia.

Uruchom ***PHPMyAdmin*** i sprawdź czy jest tam baza o nazwie ***test***. Jeżeli nie ma, to utwórz ją. Następnie dla tej bazy wykonaj zapytania:

```
CREATE TABLE `test`.`strony` (`id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,`tytuł` VARCHAR(30) NOT NULL) ENGINE = MYISAM ;
INSERT INTO `test`.`strony` (`id` ,`tytuł`) VALUES (NULL , 'A') , (NULL , 'B') ,
(NULL , 'C') , (NULL , 'D');
```

Utwórz podatny na atak skrypt o nazwie ***sql.php***, który będzie wyświetlał tytuły z bazy (lub jeden z nich po podaniu parametru ***id*** przekazanego metodą ***POST***) - skorzystaj z klasy ***Baza*** z poprzednich laboratoriów:

```
<?php
include_once 'Baza.php';
$ob = new Baza('localhost','root','','test');
if (isset($_POST['id']))
{
 echo 'Wybrany:
';
 $id=$_POST['id'];
 echo 'SQL: SELECT tytuł FROM strony WHERE id="'. $id .'
';
 echo $ob->select('SELECT id,tytuł FROM strony WHERE
 id="'. $id .'',array('id','tytuł'));
}
else
{
 echo '<form action="sql.php" method="post">';
 echo 'Wpisz numer ID do pokazania: <input type="text" name="id">';
 echo '<input type="submit" value="Uruchom">
';
 echo 'Wszystkie:
';
 echo $ob->select('SELECT id,tytuł FROM strony;',array('id','tytuł'));
}
?>
```

Przetestuj działanie skryptu - czy wyświetla poprawnie wszystkie rekordy i czy po podaniu ***id*** w polu tekstowym wyświetla właściwy rekord?

Spróbuj dokonać ataku ***SQL Injection***, który spowoduje wyświetlenie dodatkowego rekordu o ***id=2***. Wpisz do pola tekstowego następujący tekst:

1" or id="2

Spróbuj wykonać teraz atak wpisując do pola tekstowego:

1";DELETE FROM strony WHERE id="1

Prawdopodobnie nie uda się wykonać ataku z dodatkowym zapytaniem (np. kasującego rekordy), ponieważ funkcje PHP uniemożliwiają wykonanie większej liczby zapytań w jednej metodzie ***query()***.

Chcąc wykonywać wiele zapytań (co jak widać nie jest bezpieczne), należałooby użyć metody ***multi\_query()*** zamiast ***query()*** dla obiektu ***mysqli***. Nie będziemy tego testować.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

W celu zabezpieczenia skryptu przed atakami SQL Injection należy wykonać następującą modyfikację - linię:

```
$id=$_POST['id'];
```

zamienić na:

```
$id=addslashes($_POST['id']);
```

Przetestuj ponownie wyświetlanie dwóch rekordów (nie powinno już zadziałać).

Dodaj do klasy *Baza* dwie nowe metody:

1. Metodę, która będzie zabezpieczała wprowadzone dane przez użycie dedykowanej metody z klasy *mysqli*:

```
public function protect_string($str) {
 return $this->mysqli->real_escape_string($str);
}
```

2. Metodę dla danych, które powinny mieć wartości całkowite:

```
public function protect_int($nmb) {
 return (int)$nmb;
}
```

Zmień zmodyfikowaną wcześniej linię z pliku *sql.php* na:

```
$id=$ob->protect_string($_POST['id']);
```

Przetestuj ponownie atak. Możesz także użyć drugiej metody, czyli *protect\_int*, ponieważ pole *id* jest typu INT. Atak w obu przypadkach już się nie powiedzie.

### Zadanie 9.6. Bezpieczeństwo sesji w PHP

Mechanizm sesji w PHP nie jest doskonały i należy zadbać również o jego bezpieczeństwo. Wadą tego systemu jest zastosowanie identyfikatora w adresie URL. Przez odpowiednie ustawienia w pliku *php.ini* można wymusić stosowanie wyłącznie ciasteczek - za pomocą opcji *session.use\_only\_cookies* oraz *session.trans\_sid*.

Sesje mają domyślnie dość długi czas trwania, co daje atakującemu dużo czasu na przygotowanie ataku. Należy więc samodzielnie kontrolować ten czas.

W ćwiczeniu przedstawione zostanie zabezpieczenie przed atakiem przejęcia identyfikatora sesji oraz wymuszenia sesji.

Sprawdź, jaką wartość w pliku *php.ini* ma opcja *session.use\_only\_cookies*.

Jeśli jest ustalona na 1, to na czas trwania tego ćwiczenia zmień jej wartość na 0.

Przygotuj skrypt *sesja.php*:

```
<?php
 session_start();
 echo 'ID: '.session_id().'
 if (isset($_GET['PHPSESSID']))
 echo 'PHPSESSID: '.$_GET['PHPSESSID'].'
?>
```

Po uruchomieniu skryptu na stronie pojawi się wygenerowany przez serwer identyfikator sesji.

Następnie zamknij i ponownie uruchom przeglądarkę wpisując adres:

*sesja.php?PHPSESSID=123*



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

Zauważ, że identyfikator sesji zostanie teraz ustawiony na ten wpisany ręcznie (co może nie zadziałać, z uwagi na lepsze zabezpieczenia stosowane w nowszych wersjach PHP). Taki sposób dawał atakującemu możliwość przechwycenia identyfikatora zalogowanego użytkownika, a więc uzyskania jego uprawnień. Sprawdź w katalogu **xampp/tmp** czy został utworzony plik z podstawionym **id** sesji (**sess\_123**) i czy teraz zadziała próba zmiany identyfikatora sesji.

Zabezpieczenia przed taką próbą przejęcia sesji są obecnie ustawione w pliku **php.ini**, ale należy również w swoim skrypcie zwiększyć bezpieczeństwo pracy z sesją, stosując możliwość regenerowania ID sesji. W tym celu zmodyfikuj skrypt **sesja.php**.

Po **session\_start()** dodaj:

```
if (!isset($_SESSION['our_own']))
{
 session_regenerate_id();
 $_SESSION['our_own'] = true;
}
```

Po wykonaniu zadania w pliku **php.ini** ustaw ponownie:

**session.use\_only\_cookies=1**

### Zadanie 9.7. Atak typu Cross-Site Request Forgery

Atak CSRF polega na tym, że atakujący umieszcza adres do pewnego skryptu jako adres obrazka lub ramki. Skrypt ten najczęściej jest zabezpieczony przed wykonaniem dla użytkowników nieupoważnionych. Ale jeżeli obrazek lub ramka zostanie otwarta przez zalogowanego użytkownika z odpowiednimi uprawnieniami w systemie (np. administratora), to ten użytkownik może nieświadomie wykonać niebezpieczną akcję. Atak jest szczególnie niebezpieczny, jeśli parametry są przekazywane metodą GET.

Wykonanie tego ataku można uniemożliwić poprzez wysyłanie kluczowych parametrów metodą POST. Nie zawsze jest to skuteczne, ponieważ można wysyłać żądania metodą POST przy użyciu JavaScript. Dlatego nie należy odwiedzać innych stron, jeśli jesteśmy zalogowani w jakimś serwisie.

W swoich skryptach należy zabezpieczyć się przed próbą osadzenia JavaScript.

Skrypt w tym ćwiczeniu będzie używał poprzednio utworzonej tabeli **strony** w bazie danych **test**. Sprawdź, czy w klasie **Baza** posiadasz zaimplementowaną metodę **delete()**, a następnie przygotuj nowy skrypt **delete.php**.

```
<?php
include_once 'Baza.php';
$ob = new Baza('localhost','root','','test');
if (isset($_GET['id']))
 if ($ob->delete('DELETE FROM strony WHERE id=""'.
 $ob->protect_int($_GET['id']).'"'))
 echo 'Skasowano rekord o id='.$_GET['id'];
 else
 echo 'Nie można skasować rekordu!';
?>
```



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Przygotuj także plik *csrf.html* i umieść w nim:

```
<h3>Zobacz obrazek:
 </h3>
```

Uruchom plik *csrf.html* i sprawdź przy użyciu *PHPMyAdmina*, czy rekord o wskazanym *id* został usunięty.

W celu zabezpieczenia skryptu - w pliku *delete.php* zmień wszystkie odwołania GET na POST. Jeszcze raz wywołaj plik *csrf.html* - zmień wcześniej identyfikator w obrazku lub przywróć skasowany rekord.

Do pliku *csrf.html* dodaj formularz do kasowania rekordu i sprawdzić jego działanie:

```
<form action="delete.php" method="post">
 ID:<input type="number" name="id" />
 <input type="submit" value="Kasuj" />
</form>
```

Formularz symuluje przekazanie parametrów metodą POST, które można zrealizować przy pomocy JavaScript. Wtedy też możemy wykorzystać zwykłe odnośniki, które zamiast prowadzić do pliku *delete.php* prowadzą do funkcji JS, która wyśle żądanie HTTP metodą POST.

### Zadanie 9.8. Niebezpieczne funkcje PHP

Jedną z takich funkcji jest *system()*, która pozwala na wykonywanie poleceń systemowych przez skrypt PHP. Nie należy do niej przekazywać parametrów, podawanych przez użytkownika. W przypadku takiej konieczności należy przefiltrować parametry za pomocą funkcji *escapeshellcmd()*. Funkcja usuwa znaki pozwalające na wykonywanie innych komend.

Kolejną niebezpieczną funkcją jest *exec()*.

Wiele serwerów blokuje możliwość wykorzystywania tych funkcji za pomocą opcji w pliku *php.ini*:

```
disable_functions = system,exec
```

Aby przetestować funkcję *system()*, przygotuj skrypt *system.php*:

```
<?php
 system('cmd.exe /C '.$_GET['cmd']);
?>
```

Uruchom skrypt i wpisz w adresie:

```
system.php?cmd=ping localhost
```

albo:

```
system.php?cmd=type *.*
```

W ten sposób możemy wykonać dowolną komendę. Na systemach Uniksowych możemy wstawić średnik i wykonać wiele komend.

Zmodyfikuj skrypt:

```
system('cmd.exe /C '.escapeshellcmd($_GET['cmd']));
```

**Nie zabezpieczy to skryptu przed wykonaniem powyższego przykładu w systemie Windows. Dlatego lepiej nigdy nie używać tych funkcji w swoich skryptach.**

## LABORATORIUM 10. MODUŁOWY SZABLON STRONY W OPARCIU O DEFINICJE KLASY ZARZĄDZAJĄCEJ WYGLĄDEM STRONY

### Cel laboratorium:

Celem zajęć jest stworzenie modułowego szablonu strony małej firmy, z wykorzystaniem rozwiązań przygotowanych na poprzednich laboratoriach.

### Zakres tematyczny zajęć:

Wykorzystanie klas do zaprogramowania aplikacji składającej się ze stron statycznych i dynamicznych, współpracującej z danymi przechowywanymi w bazie danych.

### Pytania kontrolne:

1. Jaki jest podstawowy schemat prostej aplikacji internetowej?
2. Jakie składowe powinna zawierać klasa do zarządzania wyglądem i treścią strony?

### Zadanie 10.1. Tworzenie szablonu strony

Wykorzystaj fragmenty skryptów i klasy tworzone na poprzednich laboratoriach i zbuduj modułowy szablon prostej aplikacji internetowej, współpracujący z bazą danych *klienci* (Rys. 10.1).

W tym celu:

- a) utwórz nowy projekt PHP o nazwie np. *SzablonPHP*,
- b) do projektu dodaj foldery *zdjecia*, *miniaturki* oraz utwórz foldery o nazwie *css*, *klasy* i *skrypty* (Rys. 10.2),
- c) do folderu *klasy* skopiuj skrypty z definicją klas *Baza.php* i *Strona.php* (przeanalizuj kod z Listingiem 10.1),
- d) korzystając z klasy *Strona.php* wygeneruj zawartość pliku *index.php* (Listing 10.2) oraz w folderze *skrypty* umieść plik *glowna.php* (Listing 10.3). Skorzystaj z uproszczonego zestawu reguł CSS (Listing 10.4) i zobacz efekt w przeglądarce (Rys.10.1).

#### Listing 10.1. Klasa Strona.php

```
<?php
class Strona {
 //pola (własności) klasy:
 protected $zawartosc;
 protected $tytul = "Modułowy serwis PHP";
 protected $slowa_kluczowe = "narzędzia internetowe, php, formularz,
 galeria";
 protected $przyciski = array("Kontakt" => "?strona=index",
 "Galeria" => "?strona=galeria",
 "Formularz" => "?strona=formularz",
 "O nas"=>"?strona=onas");
 //interfejs klasy - metody modyfikujące fragmenty strony
 public function ustaw_zawartosc($nowa_zawartosc) {
 $this->zawartosc = $nowa_zawartosc;
 }
}
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
function ustaw_tytul($nowy_tytul) {
 $this->tytul = $nowy_tytul;
}
public function ustaw_slowa_kluczowe($nowe_slowa) {
 $this->slowa_kluczowe = $nowe_slowa;
}
public function ustaw_przyciski($nowe_przyciski) {
 $this->przyciski = $nowe_przyciski;
}
public function ustaw_style($url) {
 echo '<link rel="stylesheet" href="' . $url . '" type="text/css"'
/>';
}

//interfejs klasy - funkcje wyświetlające stronę
public function wyswietl() {
 $this->wyswietl_naglowek();
 $this->wyswietl_zawartosc();
 $this->wyswietl_stopke();
}
public function wyswietl_tytul() {
 echo "<title>$this->tytul</title>";
}
public function wyswietl_slowa_kluczowe() {
 echo "<meta name=\"keywords\" contents=\"$this-
>slowa_kluczowe\">";
}
public function wyswietl_menu() {
 echo "<div id='nav'>";
 foreach($this->przyciski as $nazwa => $url){
 echo ' ' . $nazwa . '';
 }
 echo "</div>";
}
public function wyswietl_naglowek() {
 ?>
 <!DOCTYPE html>
 <html>
 <head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-
scale=1.0">
 <?php
 $this->ustaw_style('css/style.css');
 echo "<title>.$this->tytul.</title></head><body>";
 ?>
}
public function wyswietl_zawartosc() {
echo "<div id='tresc'>";
 echo "<div id='nag'>";
 echo "</div>";
 echo "<div id='menu'>";
 $this->wyswietl_menu();
}


```



```
echo "</div></div>";
echo "<div id='main'>";
echo "<h1>".$this->tytul."</h1>";
echo $this->zawartosc . "</div>"; }
public function wyswietl_stopke() {
 echo '<div id="stopka"> © BP </div>';
 echo '</body></html>';
}
}
```

**Listing 10.2.** Skrypt index.php

```
<?php

require_once("klasy/strona.php");
$strona_akt = new Strona();

//sprawdź co wybrał użytkownik:
if (filter_input(INPUT_GET, 'strona')) {
 $strona = filter_input(INPUT_GET, 'strona');
 switch ($strona) {
 case 'galeria':$strona = 'galeria';
 break;
 case 'formularz':$strona = 'formularz';
 break;
 case 'onas':$strona = 'onas';
 break;
 default:$strona = 'glowna';
 }
} else {
 $strona = "glowna";
}

//dołącz wybrany plik z ustawioną zmienną $tytul i $zawartosc
$plik = "skrypty/" . $strona . ".php";
if (file_exists($plik)) {
 require_once($plik);
 $strona_akt->ustaw_tytul($tytul);
 $strona_akt->ustaw_zawartosc($zawartosc);
 $strona_akt->wyswietl();
}
```

**Listing 10.3.** Skrypt glowna.php

```
<?php
$tytul="Informacje kontaktowe";
$zawartosc=<h2>Katedra Informatyki</h2>;
$zawartosc.= "<h4>WEiI Politechnika Lubelska
";
$zawartosc.= "ul. Nadbystrzycka 36b, 20-618 Lublin
";
$zawartosc.= "tel. +48 815252046
</h4>";
$zawartosc.= "<h3>Laboratorium PAI - tworzenie modułowego serwisu WWW.
</h3>";
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



**Listing 10.4. Uproszczony plik style.css**

```
body {
 font: 300 13px/22px "Verdana";
 color: #4E4341;
 background-color: #f4f4f4;
}
h1,h2, h3, h4, #stopka {
 text-align:center;
}
h1 {
 padding:2em 0 0 0;
}
#tresc, #stopka {
 margin:auto;
 width:90%;
 padding: 1em;
 background: #f0f0f0;
}
img {
 max-width: 100%;
 height: auto;
}

/*nawigacja 2*/
#nav{ float:left;
 width:100%;
 background:#4E4341 /*#D4816F*/ ;
 color:#f0f0f0;
}

#nav a {
 float:left;
 display:block;
 width:23%;
 padding:1%;
 font-size:120%;
 text-align:center;
 text-decoration: none;
 color:#fff;
 border-bottom: 1px solid #FFF
}

/* Responsywnoś */
@media screen and (max-width: 720px) {
 #nav a { width: 50%;
 padding: 10px 0;
 font-size:120%;
 }
}

@media screen and (max-width: 480px) {
```



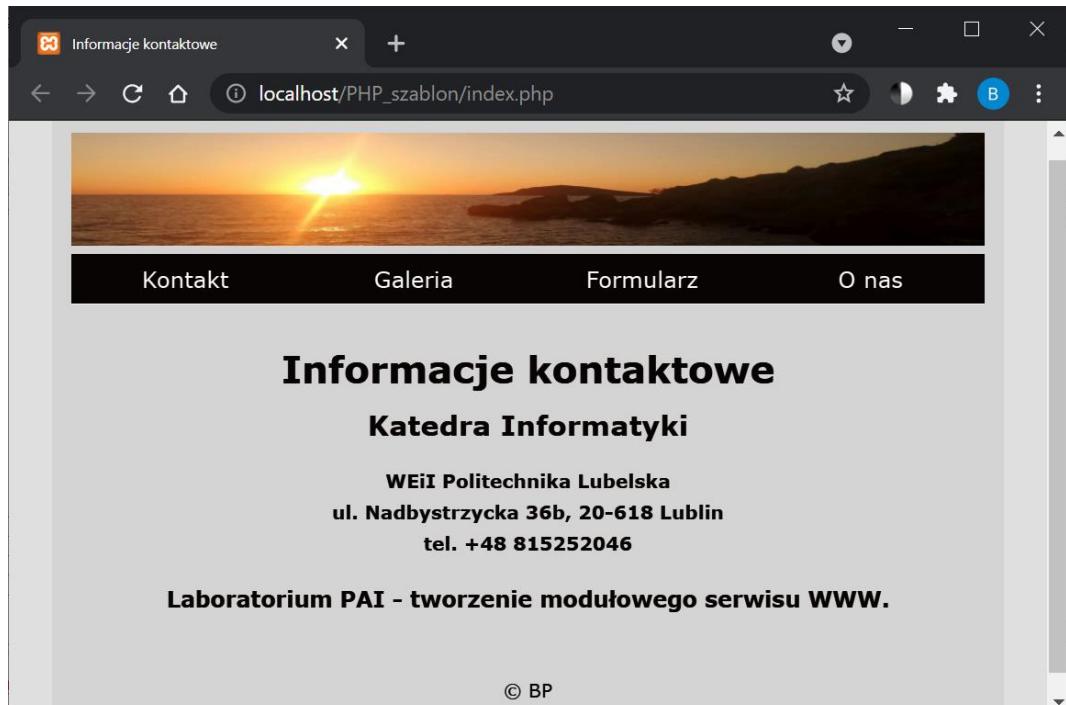
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



```
body { font-size: 14px; }
h1 { font-size: 24px; }
h2 { font-size: 18px; }
#nav a { width: 96%;
 padding: 10px 0;
 font-size: 150%; }
}
```



Rys. 10.1. Wynik działania skryptu index.php

### Zadanie 10.2. Kolejne strony serwisu

Utwórz kolejne pliki (w folderze *skrypty*) *galeria.php*, *formularz.php*, *onas.php* (w formularzu zrealizuj obsługę bazy danych korzystając z klasy *Baza.php* – Listing 10.5, 10.6). Zwróć uwagę, że w każdym z tych plików należy utworzyć dwie zmienne: *\$tytuł* i *\$zawartosc*, które są wykorzystywane do modyfikacji treści żądanej strony w skrypcie *index.php*.

Strukturę całego projektu przedstawia Rys. 10.2, a początkowy widok galerii – Rys. 10.3. Rysunek 10.4 pokazuje stronę z formularzem, opracowanym na poprzednich laboratoriach. Wszystkie akcje w tym formularzu powinny działać tak jak poprzednio (zapis danych do bazy po walidacji, pokazywanie wszystkich zamówień z bazy, pokazywanie wybranych zamówień itp.).



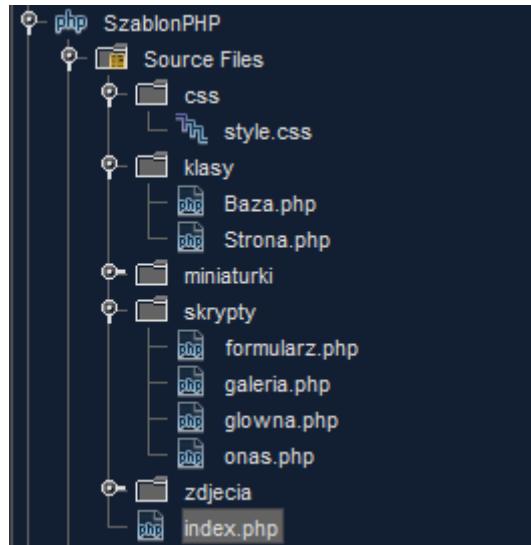
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny



Rys. 10.2. Struktura projektu

Listing 10.5. Schemat pliku galeria.php

```
<?php
$tytul = "Galeria";
$zawartosc = "Wyświetl zdjęcia z folderu z miniaturkami";
```

Listing 10.6. Schemat pliku formularz.php

```
<?php
//wykorzystaj lekko zmodyfikowane wcześniej tworzone funkcje
//pomocnicza funkcja generująca formularz:
function drukuj_form() {
 $zawartosc = '<div id="tresc">
 <form action="?strona=formularz" method="POST" >
 <table> ...
 </form>
 </div> ';
 return $zawartosc;
}

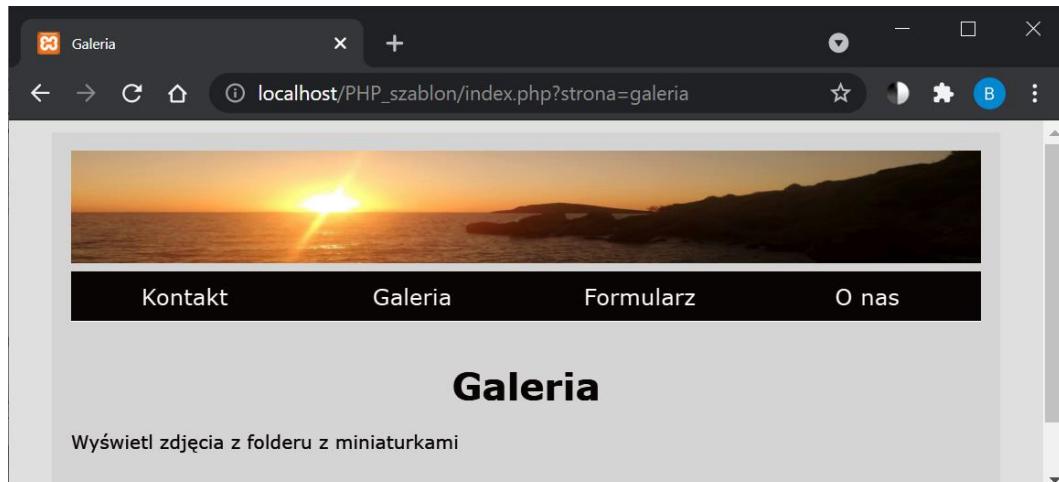
function walidacja() { //bez zmian
}
function dodajdoBD($bd) { //bez zmian
}

//uchwyt do bazy klienci:
include_once "klasy/Baza.php";
$tytul = "Formularz zamówienia";
$zawartosc = drukuj_form();
$bd = new Baza("localhost", "root", "", "klienci");

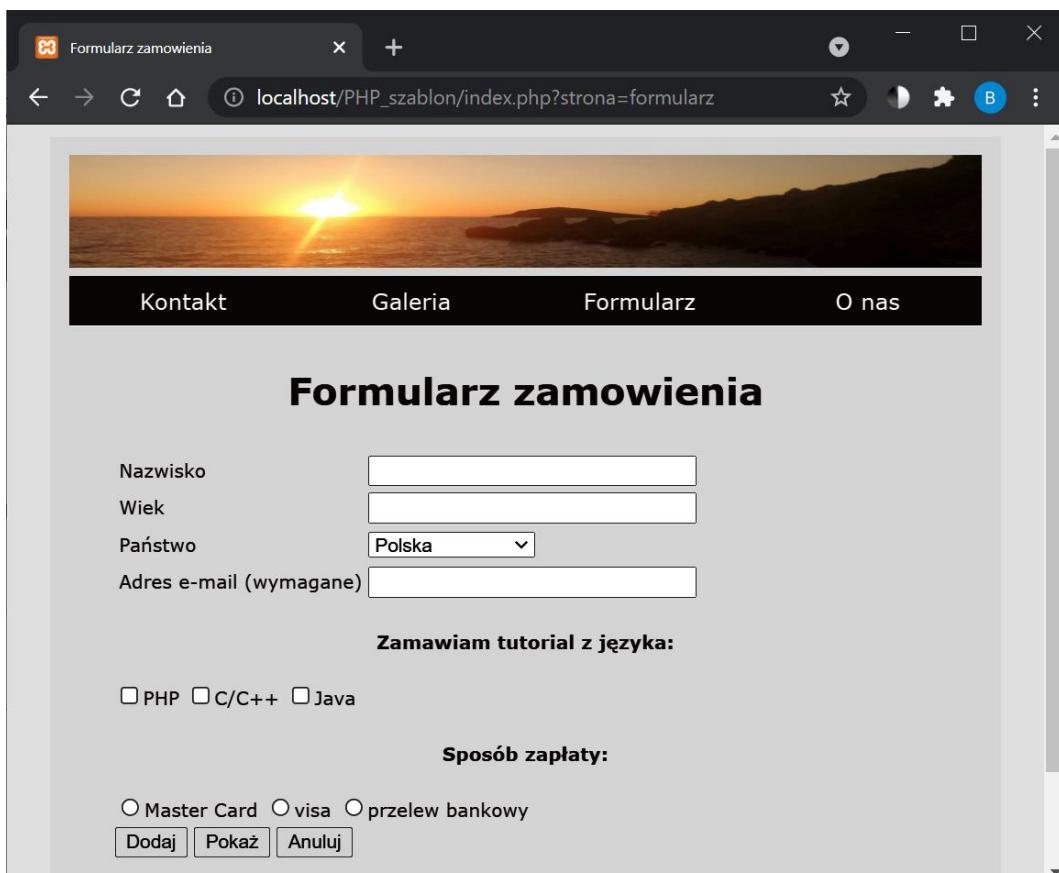
if (filter_input(INPUT_POST, "submit")) {
 $akcja = filter_input(INPUT_POST, "submit");
 switch ($akcja) {
 case "Dodaj" : dodajdoBD($bd);
```



```
 break;
 case "Pokaż" : $zawartosc.= $bd->select("select * from klienci",
 ["Email", "Zamowienie"]);
 break;
}
}
```



Rys. 10.3. Początkowy widok strony z galerią



Rys. 10.4. Strona z formularzem



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## LABORATORIUM 11. ASYNCHRONICZNA TRANSMISJA DANYCH Z SERWERA Z WYKORZYSTANIEM PHP, AJAX I JQUERY

### Cel laboratorium:

Celem zajęć jest poznanie możliwości wywoływania skryptów PHP w trybie asynchronicznym za pomocą funkcji JavaScript i biblioteki jQuery.

### Zakres tematyczny zajęć:

Obsługa żądań w trybie asynchronicznym - AJAX i interfejs Fetch API w JavaScript i jQuery. Modyfikacja projektu z laboratorium 10, polegająca na asynchronicznym wywoływaniu skryptów PHP do generowania dynamicznie treści na stronie.

### Pytania kontrolne:

1. Co to jest AJAX?
2. Do czego służy interfejs Fetch API w JavaScript?
3. Jaka jest różnica pomiędzy żądaniem przetwarzanym w trybie synchronicznym a asynchronicznym?

### Zadanie 11.1. Modyfikacja szablonu projektu z laboratorium 10

Stwórz kopię projektu z laboratorium 10, tak aby treści stron były zmieniane w tle (w trybie asynchronicznym). W tym celu należy trochę przebudować poprzedni projekt aplikacji.

W tym celu:

1. W nowym projekcie (kopii poprzedniego) należy dodać folder **js**, a w nim plik **skrypty.js**. Na razie plik **skrypty.js** pozostaw pusty.
2. W klasie **Strona** zamiast tworzyć menu w oparciu o elementy hiperłączy **<a>** - należy utworzyć przyciski i nadać im atrybuty **id** jak na Listingu 11.1.
3. Dopasować zestaw reguł CSS - odniesienia do hiperłączy w elemencie nawigacji zastąpić przyciskami.
4. W metodzie klasy **Strona** do wyświetlenia stopki, przed elementem zamkającym **</body>** należy dołączyć **skrypt.js** jak na Listingu 11.2.
5. Skrypt **index.php** uprość do postaci jak na Listingu 11.3.

#### Listing 11.1. Modyfikacje pola \$przyciski i metody wyswietl\_menu() w klasie Strona

```
protected $przyciski = array("Kontakt" => "index",
 "Galeria" => "galeria", "Formularz" => "formularz",
 "O nas"=>"onas");

public function wyswietl_menu() {
 echo "<div id='nav'>";
 foreach($this->przyciski as $nazwa => $id){
 echo " <button id='$id'> $nazwa </button>";
 }
 echo "</div>";
}
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

**Listing 11.2. Modyfikacja metody wyswietl\_stopke() klasy Strona**

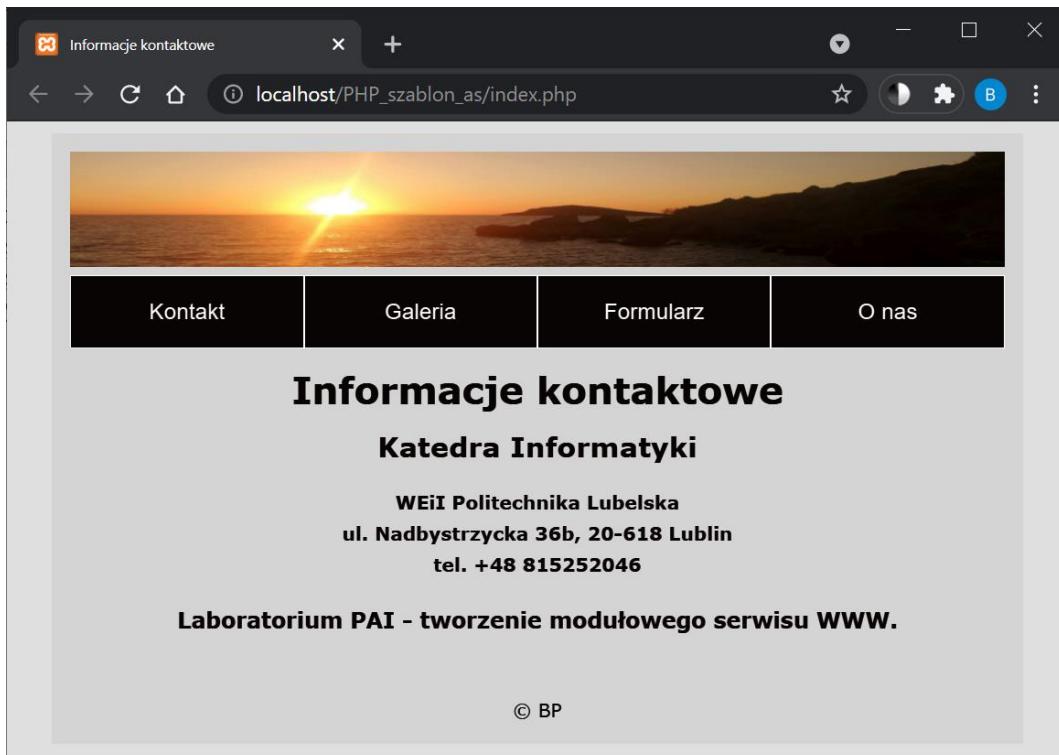
```
public function wyswietl_stopke() {

 echo '<div id="stopka"> © BP </div>';
 echo "<script src='js/skrypty.js'></script>";
 echo '</body></html>';
}
```

**Listing 11.3. Uproszczony skrypt index.php**

```
<?php
require_once("klasy/Strona.php");
$strona_akt = new Strona();
//dołącz plik z ustawioną zmienną $tytul i $zawartosc
$plik = "skrypty/glowna.php";
if (file_exists($plik)) {
 require_once($plik);
 $strona_akt->ustaw_tytul($tytul);
 $strona_akt->ustaw_zawartosc($zawartosc);
 $strona_akt->wyswietl();
}
```

Po wprowadzeniu powyższych zmian, uruchom *index.php* - efekt widoczny w przeglądarce powinien być niemal identyczny z widokiem strony głównej z poprzedniego laboratorium (Rys. 11.1), tylko teraz zamiast hiperłączy do skryptów PHP, utworzone są przyciski z *id*, które zostaną wykorzystane w dalszej części laboratorium do obsługi akcji (żądań) w trybie asynchronicznym.



Rys. 11.1. Widok w przeglądarce po wprowadzonych zmianach



Fundusze Europejskie  
Wiedza Edukacja Rozwój



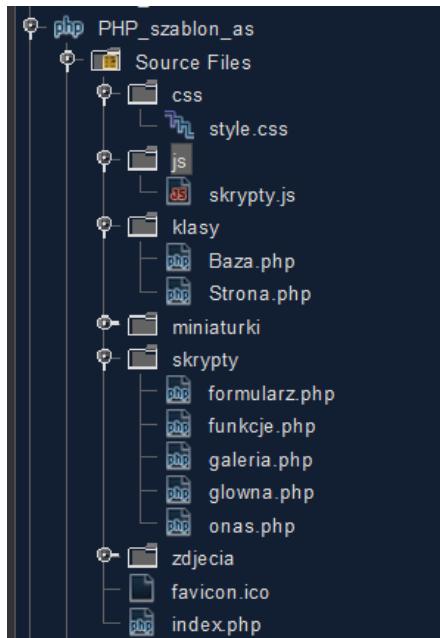
Rzeczypospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

### Zadanie 11.2. Asynchroniczna wymiana treści w szablonie - plik skrypty.js

Struktura projektu po modyfikacjach z zadania 11.1 przedstawiona jest na Rys. 11.2.



Rys. 11.2. Struktura projektu z dodatkowym folderem js i plikiem skrypty.js

- W celu przetestowania przesyłania żądań w **trybie asynchronicznym**, zmodyfikujemy teraz zawartość skryptów **galeria.php** i **onas.php**, tak, aby ich treść pobierana była za pomocą skryptu JavaScript **skrypty.js** w odpowiedzi na kliknięcie wybranego przycisku w nowej nawigacji. Przejrzyj kod źródłowy strony **index.php** i zwróć uwagę, że teraz nawigacja w HTML opiera się na 4 przyciskach z odpowiednimi **id**, jak pokazuje Listing 11.4. Kod ten jest wynikiem działania metody **wyswietl\_menu()** klasy **Strona** w skrypcie **index.php**.

#### Listing 11.4. Kod źródłowy elementu nawigacji widoczny w przeglądarce

```
<div id='nav'>
<button id='index'> Kontakt </button>
<button id='galeria'> Galeria </button>
<button id='formularz'> Formularz </button>
<button id='onas'> O nas </button>
</div>
```

- Zapoznaj się z przykładem wykładowym na temat uruchamiania skryptów PHP za pomocą AJAX i interfejsu **Fetch API**. Aby zaimplementować w bieżącym projekcie asynchronicznie wymienianą treść w elemencie **<div id='main'>** (sprawdź w kodzie źródłowym zawartość tego elementu, utworzoną przez metodę **wyswietl\_zawartosc()** klasy **Strona**), należy do pliku skrypty.js dodać funkcje jak na Listingu 11.5.

#### Listing 11.5. Plik skrypty.js

```
//Fetch API
document.addEventListener('DOMContentLoaded', () => {
 var bonas = document.getElementById('onas');
```



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



```
bonas.addEventListener("click", () => {
 console.log("Strona 0 nas");
 pokazOnas();
});
var bgaleria = document.getElementById('galeria');
bgaleria.addEventListener("click", () => {
 console.log("Galeria zdjęć");
 pokazGalerie();
});
//dodaj słuchaczy akcji do pozostałych przycisków w nawigacji
// ...
});

function pokazOnas() {
 fetch("http://localhost/PHP_szablon_as/skrypty/onas.php")
 .then((response) => {
 if (response.status !== 200) {
 return Promise.reject('Coś poszło nie tak!');
 }
 return response.text();
 })
 .then((data) => {
 document.getElementById('main').innerHTML=data;
 })
 .catch((error) => {
 console.log(error);
 });
}

function pokazGalerie() {
 fetch("http://localhost/PHP_szablon_as/skrypty/galeria.php")
 .then((response) => {
 if (response.status !== 200) {
 return Promise.reject('Coś poszło nie tak!');
 }
 return response.text();
 })
 .then((data) => {
 document.getElementById('main').innerHTML=data;
 })
 .catch((error) => {
 console.log(error);
 });
}

//dodaj funkcje do obsługi kolejnych akci
```

3. Teraz pozostaje już tylko odpowiednio zmodyfikować skrypty *onas.php* i *galeria.php*, jak pokazano na Listingach 11.6. i 11.7.



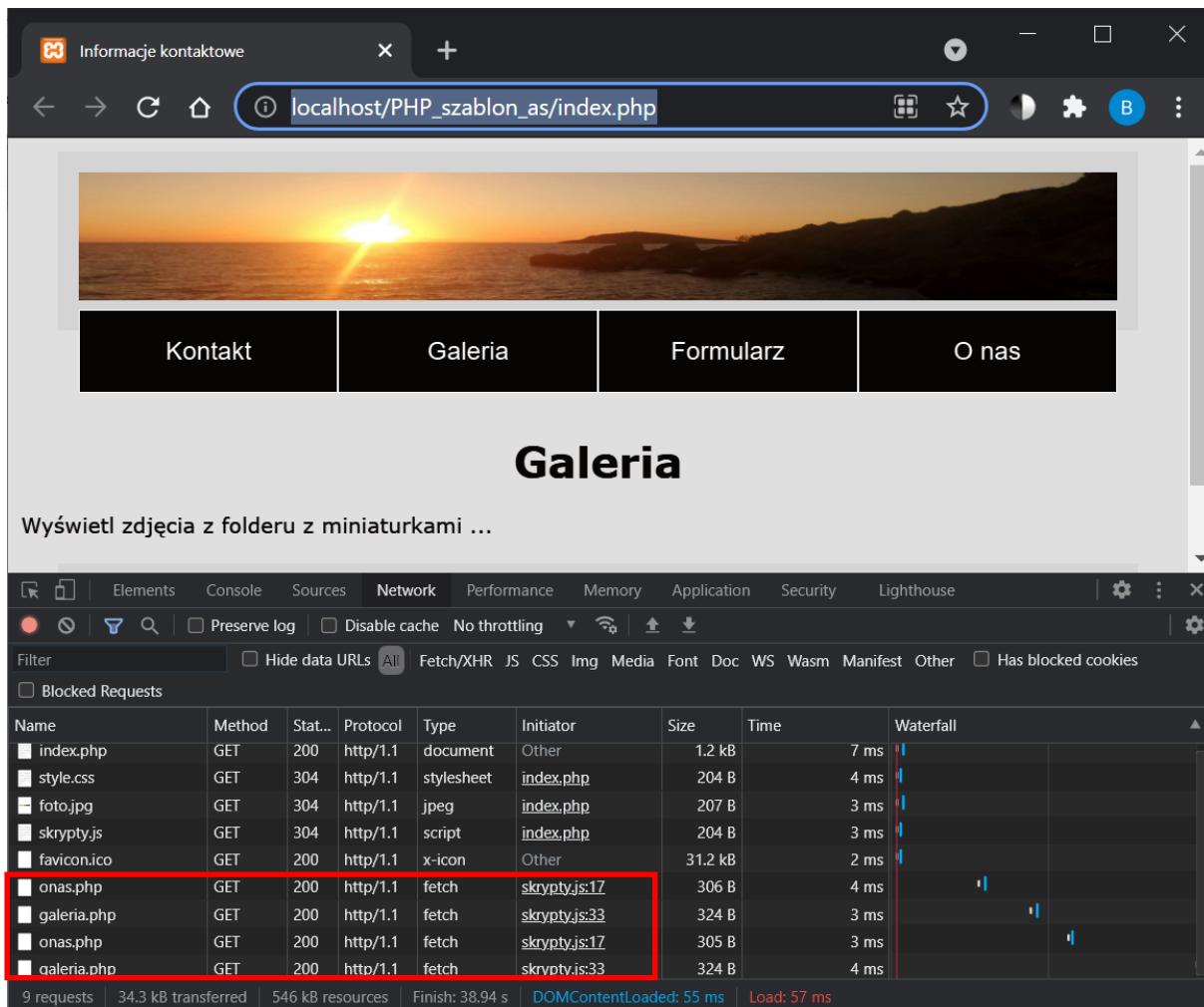
**Listing 11.6. Plik onas.php**

```
<?php
echo "<h1>O nas</h1>";
echo "<p>Informacje o naszej stronie ... </p>";
```

**Listing 11.7. Plik galeria.php**

```
<?php
echo "<h1>Galeria</h1>";
echo "<p>Wyświetl zdjęcia z folderu z miniaturkami ... </p>";
```

- Uruchom **index.php** w przeglądarce z aktywną zakładką **Network** na pasku dla deweloperów (Rys. 11.3) i przetestuj działanie aplikacji. Sprawdź, które żądania są obsługiwane w trybie asynchronicznym za pomocą **Fetch API**, a które synchronicznie. Obserwuj też czy zmienia się adres URL w pasku adresowym przeglądarki.



Rys. 11.3. Lista przykładowych żądań

- Zadania zrealizowane w punktach 1-4 można zrobić także w prostszy sposób, pomijając klasę **Strona**, podobnie jak w przykładzie wykładowym. Samodzielnie poeksperymentuj ze skryptem **formularze.php**.
- Zrealizuj te same działania, ale zamiast FetchAPI zastosuj funkcję **ajax** z biblioteki jQuery (wykład).



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## LABORATORIUM 12. CZĘŚĆ 1. IMPLEMENTACJA WZORCA PROJEKTOWEGO MVC W PRZYKŁADOWEJ APLIKACJI TYPU CRUD

### Cel laboratorium:

Celem zajęć jest przygotowanie aplikacji w oparciu o wytyczne wzorca projektowego MVC i poznanie podstawowych elementów implementacji tego wzorca w języku PHP na przykładzie szkieletu programistycznego Laravel.

### Zakres tematyczny zajęć:

Poznanie podstawowych elementów wzorca MVC na przykładzie szkieletu programistycznego Laravel w PHP. Utworzenie prostej aplikacji z autoryzacją użytkowników do zarządzania komentarzami. Obsługa akcji dodawania do bazy i pobierania danych z bazy.

### Pytania kontrolne:

1. Co to są wzorce projektowe?
2. Wyjaśnij skrót MVC i scharakteryzuj jego elementy.
3. Wyjaśnij skrót ORM. Kiedy warto skorzystać z narzędzi ORM?

### Zadanie 12.1. Przygotowanie aplikacji z autoryzacją użytkownika do obsługi akcji dodawania i wyświetlania komentarzy

Do wykonania przykładowej aplikacji skorzystamy z pakietu **XAMPP**, menedżera pakietów **Composer** oraz **Laravel** w wersji 8.

W celu utworzenia projektu Laravel potrzebujemy menedżera pakietów Composer, używanego przez nowoczesne aplikacje PHP. Composer jest używany do zarządzania zależnościami w aplikacjach i do instalowania komponentów w projektach PHP. Do pobrania ze strony <https://getcomposer.org/download/> - instalacja polega na uruchomieniu pliku *Composer-Setup.exe* (zainstalowana zostanie najnowsza wersja Composer – obecnie 2.0).

#### 1. Utworzenie pierwszej aplikacji w Laravel

W wierszu poleceń wydaj komendę, która pobierze instalator Laravel:

```
composer global require laravel/installer
```

Następnie utwórz nowy projekt Laravel w folderze *C:\xampp\htdocs*. Stworzenie nowego projektu o nazwie *tai* zrealizuj za pomocą polecenia (z poziomu katalogu, w którym ma się znaleźć projekt):

```
composer create-project --prefer-dist laravel/laravel tai
```

Przejdź do folderu *tai* utworzonego projektu (będzie to nasz **główny katalog**, z którego będziemy wydawać kolejne polecenia). Za pomocą narzędzia *artisan* (interfejs wiersza poleceń dołączony do programu Laravel) uruchom serwer developerski, na którym można testować tworzoną aplikację (Rys. 12.1):

```
php artisan serve
```



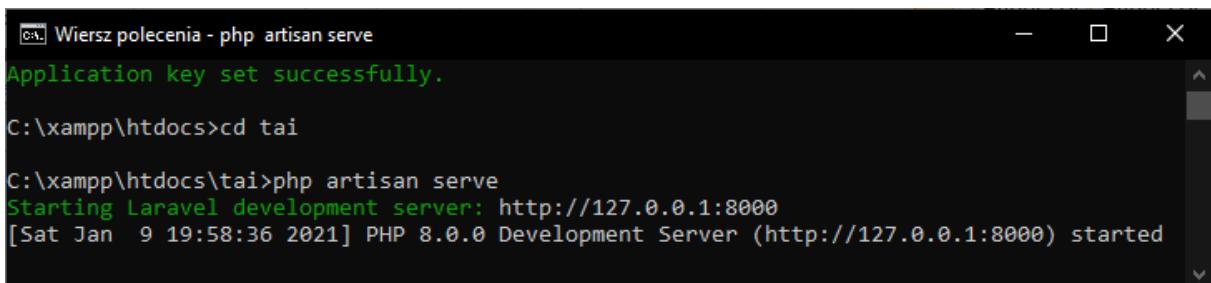
Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny



```
Wiersz polecenia - php artisan serve
Application key set successfully.

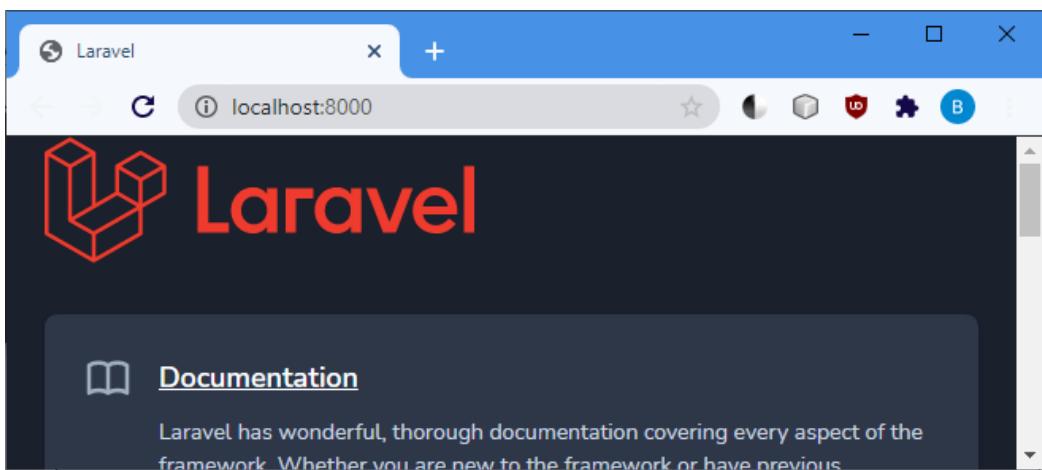
C:\xampp\htdocs>cd tai

C:\xampp\htdocs\tai>php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Sat Jan 9 19:58:36 2021] PHP 8.0.0 Development Server (http://127.0.0.1:8000) started
```

Rys. 12.1. Polecenie uruchomienia serwera deweloperskiego

Uruchom przeglądarkę i przejdź pod adres **localhost:8000**. Widok jak na Rys. 12.2 oznacza, że instalacja przebiegła pomyślnie i utworzony został pierwszy projekt Laravel o nazwie **tai**.

Korzystając z serwera deweloperskiego za każdym razem w celu wydania nowej komendy trzeba stopować serwer, wydać komendę i ponownie uruchamiać serwer do testowania aplikacji.

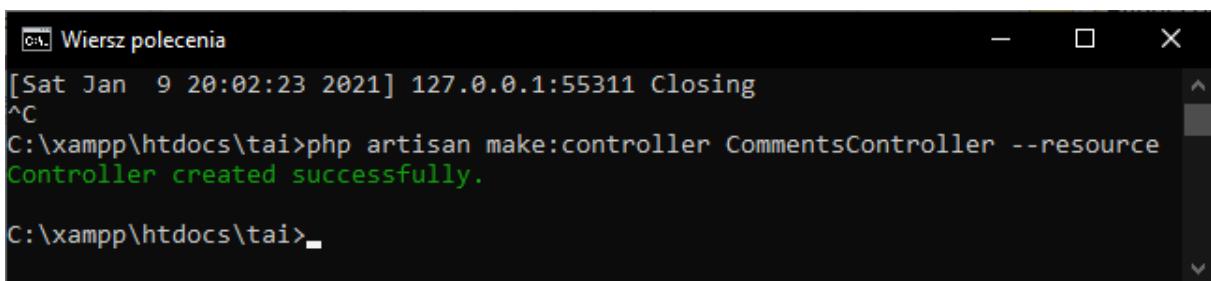


Rys. 12.2. Widok po stworzeniu projektu laravel w przeglądarce

## 2. Pierwszy kontroler, routing i widok

W programowaniu webowym, **routing** (inaczej trasowanie) jest silnikiem napędowym każdej witryny czy aplikacji. Podstawowy routing służy do kierowania żądania do odpowiedniego kontrolera. W celu zobrazowania jak działa routing, utwórz pierwszy **kontroler**. W tym celu przejdź do wiersza poleceń (zastopuj serwer deweloperski - **Ctrl+C**) i ponownie korzystając z narzędzia **artisan** w katalogu z projektem wydaj polecenie (Rys. 12.3):

```
php artisan make:controller CommentsController --resource
```



```
Wiersz polecenia
[Sat Jan 9 20:02:23 2021] 127.0.0.1:55311 Closing
^C
C:\xampp\htdocs\tai>php artisan make:controller CommentsController --resource
Controller created successfully.

C:\xampp\htdocs\tai>
```

Rys. 12.3. Polecenie tworzenia kontrolera



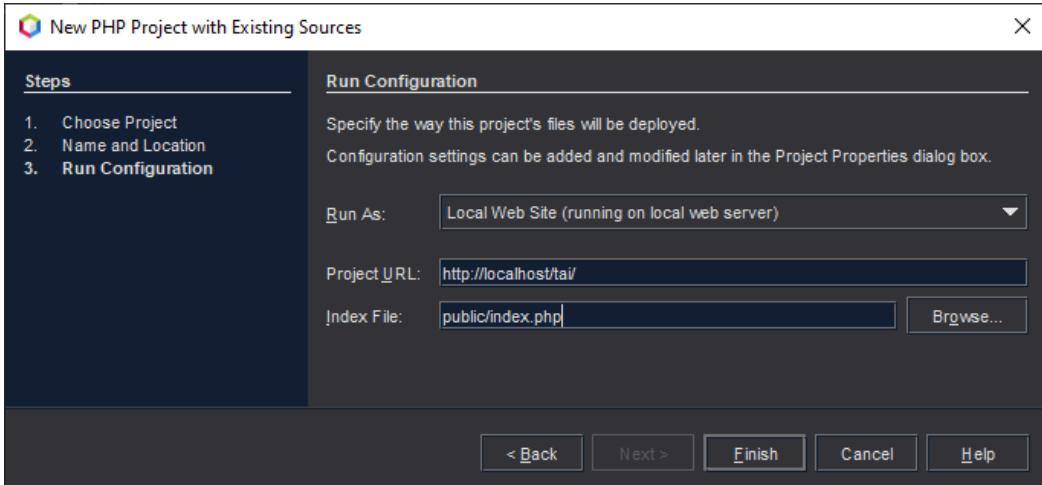
Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Do dalszej pracy z projektem dobrze jest skorzystać ze środowiska programistycznego. Na przykład, korzystając z NetBeans, utwórz nowy projekt **PHP Application (with existing sources)** jak pokazano na Rys. 12.4 i ustaw startowy plik na **index.php** z katalogu **public** projektu **tai**.



Rys. 12.4. Utworzenie i konfiguracja projektu *tai* w NetBeans

Przejrzyj zawartość folderu **app\Http\Controllers**, gdzie został utworzony plik **CommentsController.php**, który zawiera automatycznie wygenerowany kod klasy kontrolera i kilka użytecznych (na razie pustych), odpowiednio nazwanych metod. Metody te będziemy uzupełniać w kolejnych punktach. Na początek w wygenerowanej tam metodzie **index()** dodaj instrukcję (jak pokazano na Rys. 12.5):

```
return "Hello Laravel";
```

```
class CommentsController extends Controller
{
 /**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
 public function index()
 {
 return "Hello Laravel";
 }
}
```

Rys. 12.5. Metoda *index()* w utworzonym kontrolerze *CommentsController*

Teraz do pliku **routes/web.php** dodamy regułę routingu, która określi, że żądanie o URL: **localhost:8000/comments** powinno być obsłużone przez metodę **index()** kontrolera **CommentsController** (Rys. 12.6):

```
Route::get('/comments',[CommentsController::class,'index']);
```

Zwróć uwagę na konieczność importu klasy **CommentsController** (Rys. 12.6).



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



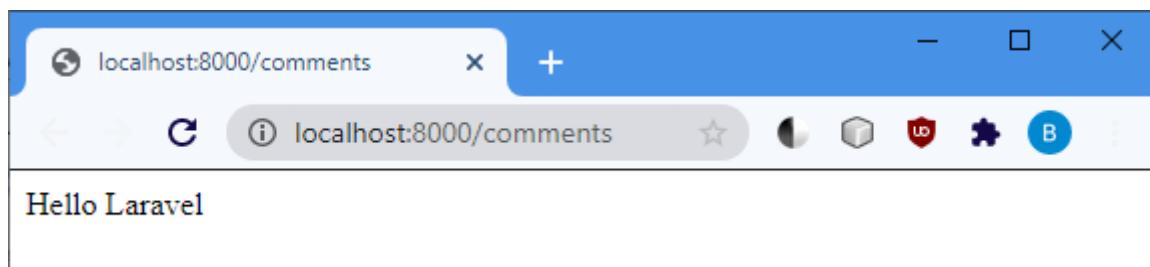
```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\CommentsController;
/*
|--
| Web Routes
|--
|
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
 return view('welcome');
});
Route::get('/comments', [CommentsController::class, 'index']);
```

Rys. 12.6. Reguły routingu w pliku routes/web.php

Przetestuj działanie nowej reguły routingu w przeglądarce - uruchom stronę <http://localhost:8000/comments> (Rys. 12.7). Pamiętaj o ponownym uruchomieniu serwera developerskiego za pomocą: php artisan serve.



Rys. 12.7. Efekt działania metody index kontrolera CommentsController

### 3. Autoryzacja użytkowników i pierwsza migracja

Laravel pozwala na bardzo prostą implementację autoryzacji użytkowników. W tym celu w wierszu poleceń w katalogu projektu należy wydać kolejne dwa polecenia (Rys. 12.8):

```
composer require laravel/ui --dev
php artisan ui vue --auth
```

A screenshot of a terminal window titled 'Wiersz polecenia'. It shows the command 'php artisan ui vue --auth' being run, followed by output indicating successful scaffolding installation and authentication generation. The path 'C:\xampp\htdocs\tai>' is visible at the bottom.

Rys. 12.8. Polecenia do autoryzacji użytkowników



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój

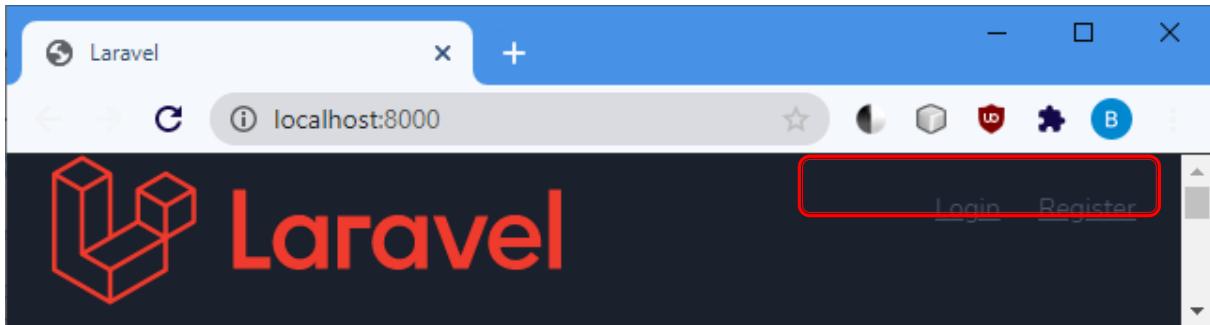


Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

Po wykonaniu tych poleceń, na stronie głównej pojawiły się opcje rejestracji i logowania jak na Rys. 12.9.



Rys. 12.9. Przyciski do rejestracji i logowania użytkownika

Aby poprawnie działało uwierzytelnienie użytkownika, należy wykorzystać (utworzone już w tym celu przez Laravel) elementy związane z utrwaleniem danych użytkownika w bazie danych. W celu ustawienia połączenia do bazy danych, w pliku `.env` (w głównym folderze projektu) należy podać dane autoryzujące dostęp do bazy danych. W przypadku MySQL domyślne ustawienia w pliku `.env` są wystarczające (Rys. 12.10).

```
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:DBx6bkY4tIGyxZYvFPuV+YZoUTzr5su37irhMCQCYQg=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8
9 DB_CONNECTION=mysql
10 DB_HOST=127.0.0.1
11 DB_PORT=3306
12 DB_DATABASE=laravel
13 DB_USERNAME=root
14 DB_PASSWORD=
```

Rys. 12.10. Domyślne ustawienia połączenia do bazy o nazwie laravel na serwerze MySQL

Korzystając z narzędzia **phpMyAdmin** utwórz nową bazę danych o nazwie **laravel** (taka nazwa bazy podana jest w domyślnym ustawieniu **DB\_DATABASE=laravel** (Rys. 12.10)).

Do pracy z bazą danych wykorzystamy tzw. migracje. **Migracje** to pliki wykonujące określone operacje na bazie danych, takie jak np. tworzenie tabel. **Pliki migracji** znajdują się w katalogu **database/migrations**. Laravel zadbał już o to, by utworzyć odpowiedni plik migracji dla tabeli użytkowników **users**. Gotowe pliki migracji można podejrzeć w projekcie (Rys. 12.11). Migracja z Rys. 12.11 tworzy tabelę **users**. Nazwy i typy pól tabeli (kolumn) zdefiniowano bezpośrednio z poziomu kodu w metodzie **up()** klasy o nazwie **CreateUserTable**, która dziedziczy po klasie **Migration**.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
 /**
 * Run the migrations.
 *
 * @return void
 */
 public function up()
 {
 Schema::create('users', function (Blueprint $table) {
 $table->bigIncrements('id');
 $table->string('name');
 $table->string('email')->unique();
 $table->timestampe('email_verified_at')->nullable();
 $table->string('password');
 $table->rememberToken();
 $table->timestamps();
 });
 }
}
```

Rys. 12.11. Gotowa klasa migracji dla tabeli Users

W celu fizycznego utworzenia tabeli *users* w bazie danych *laravel*, wystarczy już tylko wywołać komendę migracji z poziomu wiersza poleceń:

```
php artisan migrate
```

Sprawdź w *phpMyAdmin* jakie tabele zostały utworzone po uruchomieniu migracji. Zarejestruj nowego użytkownika i sprawdź efekt logowania. Przejrzyj jakie rekordy zostały dodane do tabeli *users* i *migrations*.

#### 4. Widoki dla komentarzy, formularz rejestracji

Kolejny etap to utworzenie widoków. **Widoki** są zwracane zwykle jako wynik działania metody (akcji) kontrolera. W przykładzie utworzymy stronę widoku z formularzem do wprowadzenia komentarza przez użytkownika do księgi gości.

Zadanie podzielimy na dwa etapy:

- utworzenie widoku dla księgi gości,
- utworzenie logiki dodawania komentarzy przez użytkowników.

Otwórz plik kontrolera *CommentsController*, i zmodyfikuj jego metodę *index()* która zwracała prosty napis „Hello Laravel”. Teraz zmienimy to tak, aby zwracany był specjalny **widok** za pomocą funkcji *view()*. W tym celu zmień instrukcję *return* w metodzie *index()* kontrolera *CommentsController* jak pokazano na Rys. 12.12.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

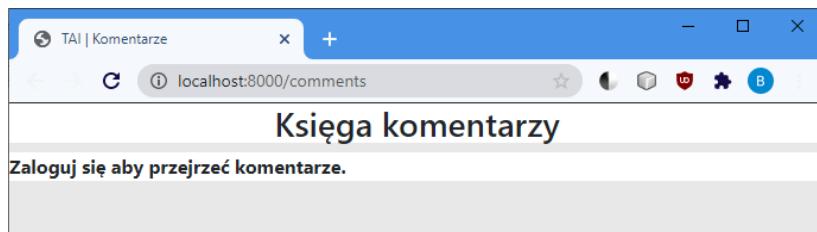


Unia Europejska  
Europejski Fundusz Społeczny

```
class CommentsController extends Controller
{
 /**
 * Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
 public function index()
 {
 return view('comments');
 }
}
```

Rys. 12.12. Modyfikacja metody index() kontrolera CommentsController

Następnie, w katalogu **resources/views** utwórz plik widoku **comments.blade.php** z treścią jak w załączonym do paczki z ćwiczeniem pliku. W Laravel widoki (dokumenty o strukturze HTML) są obsługiwane przez silnik Blade. Finalna postać pliku będzie korzystała z silnika Blade do generowania dynamicznie zawartości na podstawie danych przesyłanych z metod kontrolera. Na razie pokażemy statyczną zawartość komentarza testowego. Uruchom widok dla komentarzy '**comments**' w przeglądarce (Rys. 12.13).



Rys. 12.13. Widok w przeglądarce strony **comments.blade.php**

Laravel pozwala łatwo określić sekcje na stronie, które mają być dostępne **tylko dla zalogowanych użytkowników**. Do tego celu w pliku **comments.blade.php** dodano odpowiednie adnotacje **@auth** tuż przed widokiem dla zalogowanych użytkowników oraz **@endauth** tuż po sekcji widocznej dla zalogowanych. W adnotacjach **@guest** i **@endguest** można z kolei ustawić treść widoczną dla wszystkich użytkowników, np. z informacją o konieczności zalogowania (Rys. 12.14).

W celu założenia konta użytkownika korzystaliśmy z gotowego formularza rejestracji, który jest dostępny pod adresem <http://localhost:8000/register> (można nadać mu własny styl w odpowiednim pliku Blade, co będzie zrealizowane na kolejnym laboratorium). Mając zarejestrowanego użytkownika, można się już zalogować i przetestować ponownie działanie strony **comments**. Tym razem efekt powinien być taki jak na rys. 12.15.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny

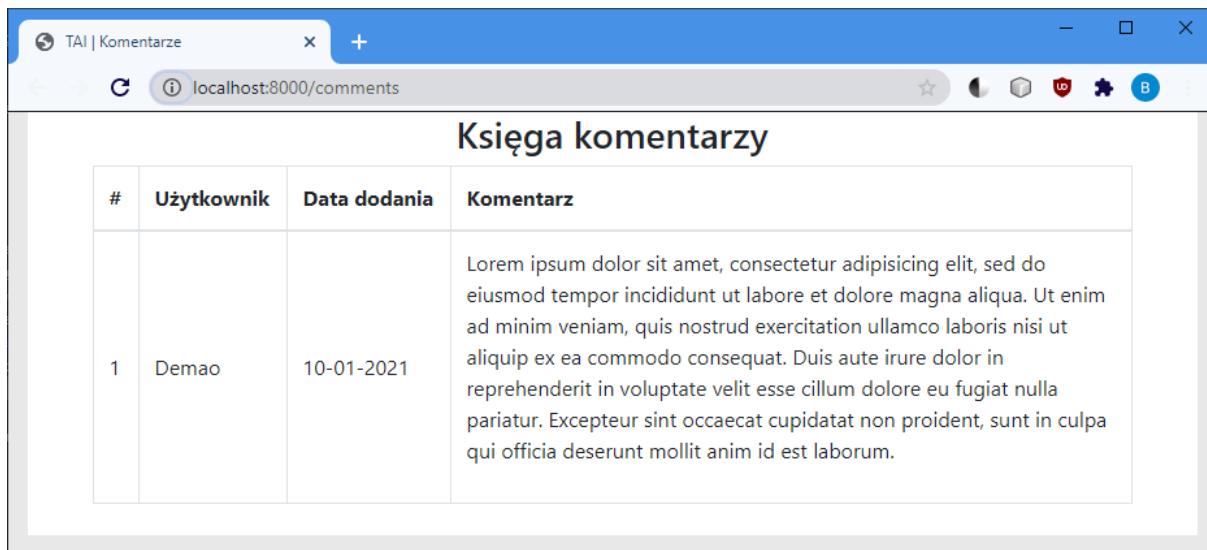


```
<body>
 <div class="table-container">
 <div class="title">
 <h3>Księga komentarzy</h3>
 </div>
 @auth
 <table data-toggle="table">
 <thead>
 <tr><...6 lines /></tr>
 </thead>
 <tbody><...7 lines /></tbody>
 </table>

 <div class="footer-button">
 Dodaj
 </div>
 @endauth
 </div>

 @guest
 <div class="table-container">
 Zaloguj się aby przejrzeć komentarze.
 </div>
 @endguest
</body>
```

Rys. 12.14. Autoryzacja dostępu do elementów strony



Rys. 12.15. Widok w przeglądarce strony **comments.blade.php**

Silnik widoków Blade oferuje wiele użytecznych metod szybkiego generowania widoków, co pozwala na łatwe tworzenie treści bezpośrednio w dokumencie HTML za pomocą różnych instrukcji sterujących, np. **@if @endif**, **@foreach @endforeach**. Te możliwości wykorzystamy w dalszych etapach ćwiczenia do pobrania i wyświetlenia komentarzy zapisanych w bazie danych.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

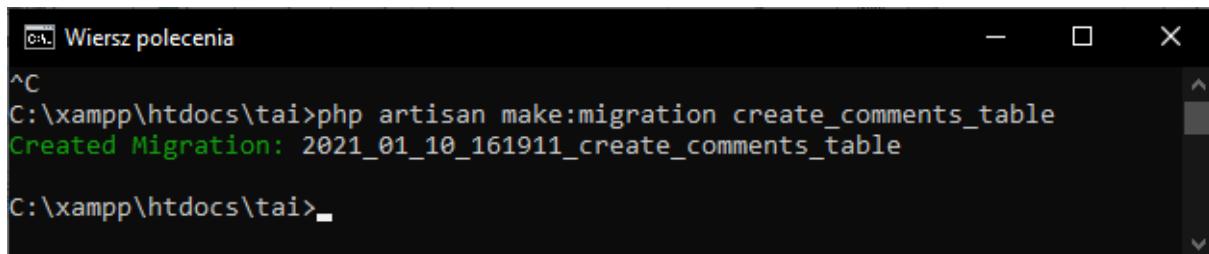


Unia Europejska  
Europejski Fundusz Społeczny

## 5. Migracja tabeli dla komentarzy. Model Comments

Potrzebujemy jeszcze tabeli w bazie danych, w której będą przechowywane komentarze użytkowników. W tym celu ponownie skorzystamy z odpowiedniej migracji. W wierszu poleceń, w katalogu projektu wpisz komendę (Rys. 12.16):

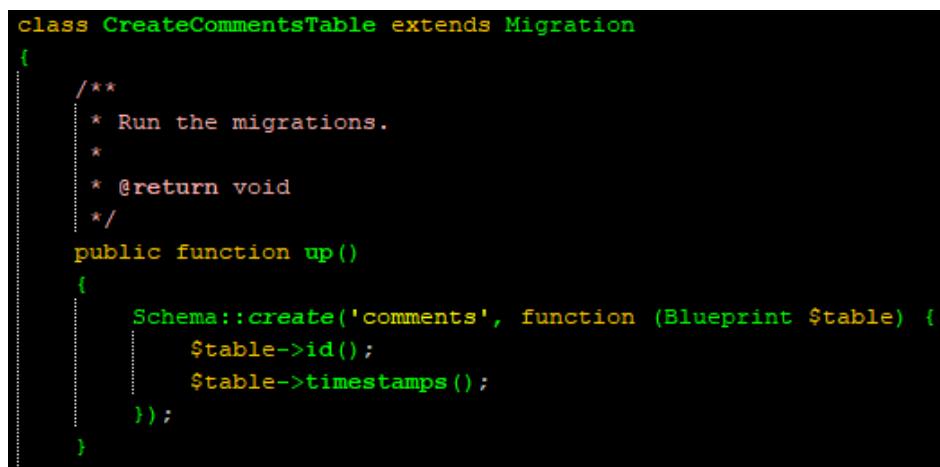
```
php artisan make:migration create_comments_table
```



```
Wiersz polecenia
^C
C:\xampp\htdocs\tai>php artisan make:migration create_comments_table
Created Migration: 2021_01_10_161911_create_comments_table
C:\xampp\htdocs\tai>
```

Rys. 12.16. Polecenie tworzenia migracji

Przejrzyj zawartość nowej migracji, która znajduje się w katalogu *database/migrations* (Rys. 12.17). Warto zauważyć, że Laravel „**odczytał**” słowa kluczowe „*create*” i „*table*” z nazwy migracji oraz wykorzystał je do dodania podstawowych właściwości (klucza głównego *id* oraz daty dodania i aktualizacji komentarza). Na podstawie nazwy migracji, stworzona tabela nazywać się będzie *comments*.



```
class CreateCommentsTable extends Migration
{
 /**
 * Run the migrations.
 *
 * @return void
 */
 public function up()
 {
 Schema::create('comments', function (Blueprint $table) {
 $table->id();
 $table->timestamps();
 });
 }
}
```

Rys. 12.17. Klasa migracji dla tabeli comments

W klasie *CreateCommentsTable*, w metodzie *up()* potrzebujemy jeszcze dwóch kolumn dla tworzonej tabeli - *treści komentarza* oraz *id użytkownika*, które za pomocą relacji 1:1 połączymy z tabelą *users*. W Laravel można to bardzo prosto zrealizować.

Dodaj najpierw brakujące pola w funkcji *up()* (Listing 12.1 i Rys. 12.18).

### Listing 12.1. Dodatkowe kolumny w tabeli comments

```
$table->bigInteger('user_id')->unsigned();
$table->text('message');
$table->foreign('user_id')->references('id')->on('users')
 ->onDelete('cascade');
```



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

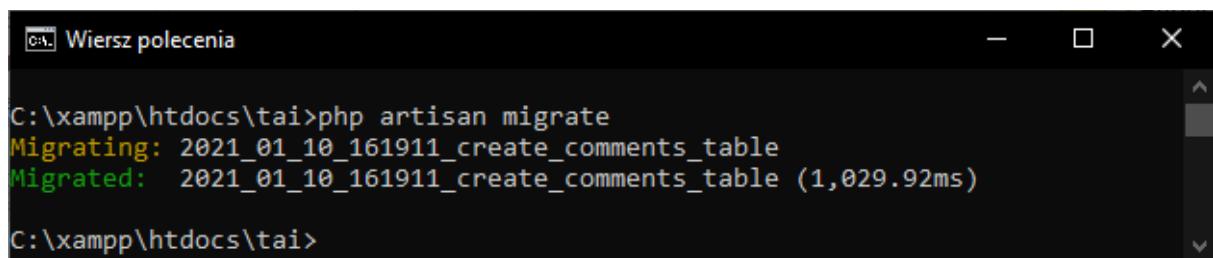


Unia Europejska  
Europejski Fundusz Społeczny

```
public function up()
{
 Schema::create('comments', function (Blueprint $table) {
 $table->id();
 $table->bigInteger('user_id')->unsigned();
 $table->text('message');
 $table->foreign('user_id')->references('id')
 ->on('users')->onUpdate('cascade')->onDelete('cascade');
 $table->timestamps();
 });
}
```

Rys. 12.18. Dodanie brakujących pól w metodzie **up()**

I wykonaj migrację jak na Rys. 12.19.



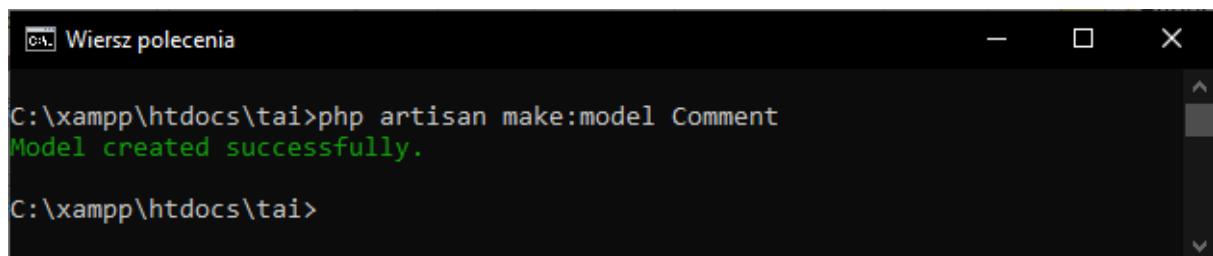
```
C:\xampp\htdocs\tai>php artisan migrate
Migrating: 2021_01_10_161911_create_comments_table
Migrated: 2021_01_10_161911_create_comments_table (1,029.92ms)

C:\xampp\htdocs\tai>
```

Rys. 12.19. Wykonanie migracji

Sprawdź w *PhpMyAdmin* efekt wykonania polecenia z Rys. 12.19. Następnie utwórz klasę modelu **Comment** (zwróć uwagę na liczbę pojedynczą w nazwie modelu), który będzie abstrakcyjną reprezentacją bytu bazodanowego dla tabeli komentarzy z poziomu aplikacji (Rys. 12.20):

```
php artisan make:model Comment
```



```
C:\xampp\htdocs\tai>php artisan make:model Comment
Model created successfully.

C:\xampp\htdocs\tai>
```

Rys. 12.20. Polecenie tworzenia modelu

W wyniku wykonania polecenia z Rys. 12.20, w katalogu **app/Models** utworzył się plik **Comment.php** z definicją klasy **Comment**. Dodaj do niego fragment (Rys. 12.21) definiujący relację jeden do jednego (jeden komentarz ma jednego autora).



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
 //relacja 1:1 dla Comments i User:
 public function user() {
 return $this->belongsTo(User::class);
 }
 use HasFactory;
}
```

Rys. 12.21. Dodanie relacji 1:1 do modelu *Comments*

Aby widok mógł wyświetlić dane z bazy danych, wykorzystamy akcję kontrolera i instrukcję, która wykona pobranie wszystkich rekordów z tabeli *comments*. W kontrolerze *CommentsController* zainportuj klasę modelu *Comment* (Rys. 12.22) – w nagłówku pliku *CommentsController.php*, zaraz po deklaracji przestrzeni nazw *namespace*, dodaj kod (Rys. 12.22):

```
use App\Models\Comment;
```

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Models\Comment;

class CommentsController extends Controller
```

Rys. 12.22. Dodanie importu klasy *Comment* do kontrolera *CommentsController*

Zależność ta będzie wykorzystana nieco później. Teraz zajmiemy się utworzeniem formularza dodawania komentarzy.

## 6. Formularz dodawania komentarza. Akcja create w kontrolerze.

Kolejny etap to utworzenie widoku formularza dodawania nowego komentarza. Tak jak poprzednio, by pokazać stronę (widok), należy najpierw w pliku *routes/web.php* dodać odpowiednią regułę routingu. Do obsługi żądań dodania komentarzy potrzebne będą dwie reguły routingu (Rys. 12.23), związane z obsługą metod:

- *get*, za pomocą której zostanie wyświetlony widok formularza dodawania komentarza (metoda *create()* kontrolera);
- *post*, która odbierze żądanie z parametrami wysyłane z formularza i przekaże do odpowiedniej metody kontrolera (w przykładzie będzie to metoda *store()*).

Rys. 12.23 przedstawia dodatkowe reguły routingu w pliku *web.php*. Zauważ, że reguły mają przypisaną nazwę za pomocą *->name('...')*. Nadanie nazw regułom routingu jest bardzo wygodne w późniejszym odwoływaniu się do nich.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
Route::get('/comments', [CommentsController::class, 'index'])->name('comments');
Route::get('/create', [CommentsController::class, 'create'])->name('create');
Route::post('/create', [CommentsController::class, 'store'])->name('store');
```

Rys. 12.23. Dodatkowe reguły routingu w routes/web.php

Po zaktualizowaniu reguł routingu, w metodzie `create()` kontrolera `CommentsController`, utwórz nowy obiekt `$comment` oraz dodaj przekierowanie do widoku z formularzem `commentsForm.blade.php` (Rys. 12.24). Przekazywanie zmiennych z kontrolera do widoków można realizować za pomocą drugiego parametru tablicowego funkcji `view()`. W przykładzie przekazywany jest tylko jeden element - obiekt `$comment`.

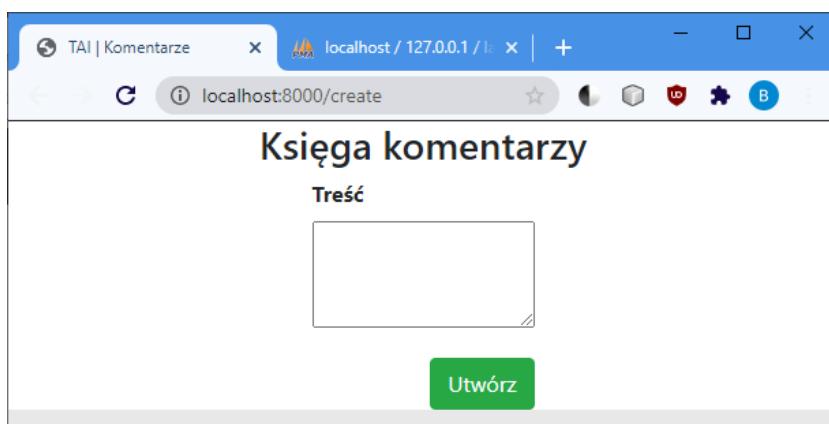
```
public function create()
{
 $comment = new Comment;
 return view('commentsForm', ['comment' => $comment]);
}
```

Rys. 12.24. Instrukcje w metodzie `create()` kontrolera `CommentsController`.

Następnie, w katalogu `resources/views` utwórz plik widoku `commentsForm.blade.php` z treścią jak w załączonym do ćwiczenia pliku (fragment pliku pokazany jest na Rys. 12.26). Na stronie widoku `commentsForm.blade.php` zwróć uwagę na obsługę błędów (zmienna `$errors` i `>{!! csrf_field() !!}`) i inne elementy specyficzne dla silnika szablonów Blade:

- instrukcje sterujące takie jak `@foreach`-`@endforeach`,
- wartość parametru `action="{!! route('/store') !!}"` w formularzu, gdzie odwołanie następuje do nazwanej reguły routingu,
- parametr `class="form-group{!! $errors->has('message')?'has-error':'' !!}"` dla elementu `div`.

Widok w przeglądarce pod adresem <http://localhost:8000/create> prezentuje Rys. 12.25.



Rys.12.25. Widok formularza dodawania komentarza



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
<div class="table-container">
 <div class="title"> <h3>Księga komentarzy</h3> </div>
 @if ($errors->any())
 <div class="alert alert-danger">

 @foreach ($errors->all() as $error)
 {{ $error }}
 @endforeach

 </div>
 @endif
 <div class="box box-primary ">
 <!-- /.box-header -->
 <!-- form start -->
 <form role="form" action="{{ route('store') }}" id="comment-form"
 method="post" enctype="multipart/form-data" >
 {{ csrf_field() }}
 <div class="box">
 <div class="box-body">
 <div class="form-group{{ $errors->has('message') ? 'has-error' : '' }}>
 <label>Treść</label>

 <textarea name="message" id="message" cols="20" rows="3" required></textarea>
 </div>
 </div>
 <div class="box-footer"><button type="submit" class="btn btn-success">Utwórz</button>
 </div>
 </div>
 </form>
 </div>

```

Rys. 12.26. Fragment pliku widoku *commentsForm.blade.php*

Aby ze strony z komentarzami przejść do formularza dodawania nowego komentarza - do pliku *comments.blade.php*, po znaczniku zamykającym tabelę *</table>*, dodaj blok *<div>* z hiperlinkiem: jak na Listingu 12.2.

#### Listing 12.2. Dodatkowy przycisk-hiperłącze na stronie *comments.blade.php*

```
<div class="footer-button">
 Dodaj
</div>
```

## 7. Walidacja formularza

Przed zapisem do bazy danych, podana treść komentarza powinna zostać sprawdzona. Można skorzystać z walidacji po stronie klienta za pomocą HTML, ale również należy walidację powtórzyć po stronie serwera. Laravel wprowadza bardzo zaawansowany moduł walidacji danych przesyłanych w żądaniach. Aby zrozumieć jak działa ten mechanizm – trzeba wiedzieć co się dzieje z danymi z formularza po ich przesłaniu w wyniku kliknięcia przycisku „Utwórz”. W pierwszej kolejności dane z formularza są wysyłane metodą POST przez protokół HTTP i odbierane przez odpowiedni kontroler Laravela w postaci obiektu **Request**. Następnie trafiają do wskazanej w regule routingu metody kontrolera (w naszym przypadku jest to metoda **store()**). Metoda **store()**, jako parametr wejściowy otrzymuje obiekt klasy **Request**. Aby przeprowadzić walidację danych przesyłanych w żądaniu należy wykorzystać funkcję **validate()**, w której można zadeklarować warunki walidacji w postaci tablicy asocjacyjnej.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



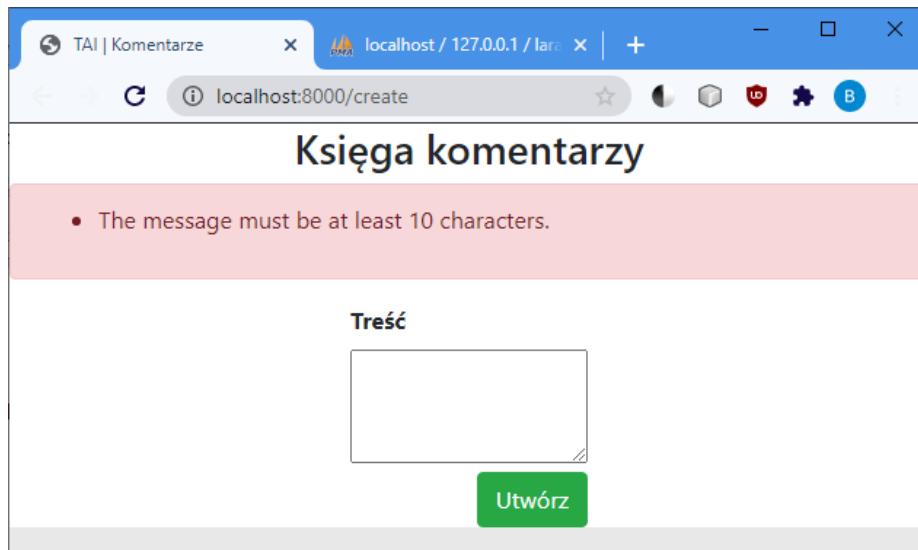
Unia Europejska  
Europejski Fundusz Społeczny

Uzupełnij metodę **store(Request \$request)** kontrolera, kodem jak na Listingu 12.3.

**Listing 12.3. Metoda store z walidacją**

```
public function store(Request $request)
{
 // Podstawowa walidacja formularza:
 $this->validate($request, [
 'message' => 'required|min:10|max:255',
]);
}
```

Przetestuj działanie validatora (Rys. 12.27).



Rys. 12.27. Działanie walidacji

## 8. Zapis komentarzy do bazy danych i wyświetlenie danych z bazy

Poprawne dane pobrane z formularza, powinny być zapisane w bazie danych. Uzupełnimy naszą metodę **store()** dodatkowymi instrukcjami jak na Rys. 12.28.

Przetestuj działanie metody **store()** - za pomocą formularza dodawania, wprowadź nowy komentarz i sprawdź w *PhpMyAdmin*, czy został on prawidłowo dodany do bazy danych.

Ostatni etap to wyświetlenie w widoku głównym wszystkich komentarzy pobranych z bazy danych. W tym celu w metodzie **index()** kontrolera **CommentsController** zmodyfikuj kod jak na Rys. 12.29. Wykorzystano tu klasę modelu **Comment**, która jest abstrakcyjnym bytem odwzorowującym zbiór bazodanowy komentarzy na kolekcję obiektów klasy **Comment** w aplikacji.

Dostęp do danych w Laravel jest najczęściej realizowany za pomocą narzędzia **Eloquent**. **Eloquent** jest warstwą ORM wykorzystywaną do mapowania obiektów aplikacji Laravel, na rekordy tabel w bazie danych (i odwrotnie). **Eloquent** pozwala przekształcić żądania obsługi danych modeli w odpowiednie zapytania i co najważniejsze, nie ogranicza się przy tym do jednego systemu bazodanowego. Metoda **store()** skorzystała z możliwości utrwalenia obiektu **Comment** w tabeli **comments** a teraz metoda **index()** (Listing 12.18) wykorzysta **Eloquent** do pobrania danych w postaci posortowanej kolekcji obiektów klasy (modelu) **Comment**.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```
public function store(Request $request) {
 //Podstawowa walidacja formularza:
 $this->validate($request, [
 'message' => 'required|min:10|max:255',
]);
 if (\Auth::user() == null) {
 return view('home'); //jesli użytkownik nie jest zalogowany
 }
 $comment = new Comment();
 $comment->user_id = \Auth::user()->id; //ID aktualnie zalogowanego usera
 $comment->message = $request->message; //Nazwa pola z validatora
 if ($comment->save()) {
 return redirect('comments');
 }
 return view('comments');
}
```

Rys. 12.28. Zapis komentarza w metodzie `store()` kontrolera `CommentsController`

```
public function index()
{
 $comments = Comment::orderBy('created_at', 'asc')->get();
 return view('comments', ['comments'=>$comments]);
}
```

Rys. 12.29. Pobranie wszystkich komentarzy z bazy danych metodą `get()`

Za pomocą metod narzędzia *Eloquent*, instrukcja:

```
Comment::orderBy('created_at', 'asc')->get();
```

zostanie przekształcona na zapytanie SQL do tabeli `comments` postaci:

```
SELECT * FROM comments ORDER BY created_at ASC
```

*Eloquent* jest narzędziem ORM (ang. Object –Relation Mapping). Jedną z jego zalet jest to, że można go wykorzystać do odczytywania wprost wartości modeli pozostających ze sobą w relacji, a co więcej nakładać na nie odpowiednie warunki. Oznacza to, że programista może pominąć żmudne i długie zapytania SQL oparte o JOIN i warunki w nich zagnieżdżone, i skorzystać z metod *Eloquent* typu: `whereHas`, `whereDoesntHave`, które pozwolą na prostsze dodanie warunków na struktury danych obiektowych. Przykładowo, jeśli zamiast wszystkich komentarzy, trzeba zwrócić tylko komentarze użytkowników, którzy mają w swoim adresie `email`, np. imię *Karolina* - to dzięki zadeklarowaniu relacji 1:1 w modelu `Comment`, wystarczy, że sformułujemy zapytanie *Eloquent* postaci:

```
$comments = Comment::whereHas('user', function ($query) {
 $query->where('email', 'like', '%karolina%');
})->get();
```

Dla porównania, czyste zapytanie SQL ma postać:

```
"select * from `comments` where exists (select * from `users` where
`comments`.`user_id` = `users`.`id` and `email` like '%karolina%'")";
```

Odczytanie danych z kolekcji komentarzy w widoku `comments.blade.php` można zrealizować za pomocą zamiany kodu w bloku `<tbody>`, jak na Rys. 12.30.



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

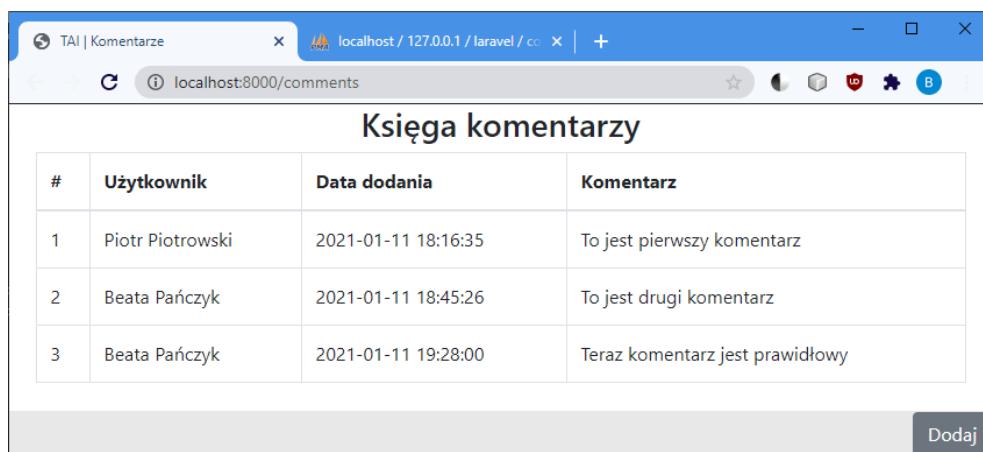


Unia Europejska  
Europejski Fundusz Społeczny

```
<tbody>
 @foreach($comments as $comment)
 <tr>
 <td>{{$comment->id}}</td>
 <td>{{$comment->user->name}}</td>
 <td>{{$comment->created_at}}</td>
 <td>{{$comment->message}}</td>
 </tr>
 @endforeach
</tbody>
```

Rys. 12.30. Zmiana zawartości bloku `<tbody>` w widoku `comments.blade.php`

Widok strony z komentarzami pobranymi z bazy danych przedstawia Rys. 12.31. Widok ten jest dostępny jedynie dla zalogowanych użytkowników.



The screenshot shows a web browser window with the title "TAI | Komentarze". The address bar indicates the URL is "localhost / 127.0.0.1 / laravel / cc" and the page is "localhost:8000/comments". The main content is a table titled "Księga komentarzy". The table has four columns: "#", "Użytkownik", "Data dodania", and "Komentarz". There are three rows of data:

| # | Użytkownik       | Data dodania        | Komentarz                       |
|---|------------------|---------------------|---------------------------------|
| 1 | Piotr Piotrowski | 2021-01-11 18:16:35 | To jest pierwszy komentarz      |
| 2 | Beata Pańczyk    | 2021-01-11 18:45:26 | To jest drugi komentarz         |
| 3 | Beata Pańczyk    | 2021-01-11 19:28:00 | Teraz komentarz jest prawidłowy |

A "Dodaj" button is visible at the bottom right of the table area.

Rys. 12.31. Widok strony z komentarzami



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska

Unia Europejska  
Europejski Fundusz Społeczny



## LABORATORIUM 12. CZĘŚĆ 2. IMPLEMENTACJA WZORCA PROJEKTOWEGO MVC W PRZYKŁADOWEJ APLIKACJI TYPU CRUD

### Cel laboratorium:

Celem zajęć jest przygotowanie aplikacji w oparciu o wytyczne wzorca projektowego MVC i poznanie podstawowych elementów implementacji tego wzorca w języku PHP na przykładzie szkieletu programistycznego Laravel.

### Zakres tematyczny zajęć:

Rozszerzenie funkcjonalności aplikacji utworzonej w zadaniu 12.1. o elementy walidacji danych z formularza oraz obsługa akcji usuwania i modyfikacji komentarzy.

### Zadanie 12.2. Dodatkowe funkcjonalności aplikacji ‘Księga komentarzy’

#### 1. Własny styl dla formularza logowania/rejestracji

Widok dla formularza logowania znajduje się w pliku **resources/views/auth/login.blade.php**. W celu dodania własnego stylu dla formularza logowania, wystarczy zmodyfikować ten plik. W pierwszej linii pliku znajduje się wyrażenie **extends**, które wykorzystuje szablon pliku **resources/layouts/app.blade.php**. Stosowanie szablonu pozwala zachować spójny wygląd całego projektu oraz umożliwia łatwe definiowanie kolejnych elementów wyglądu aplikacji.

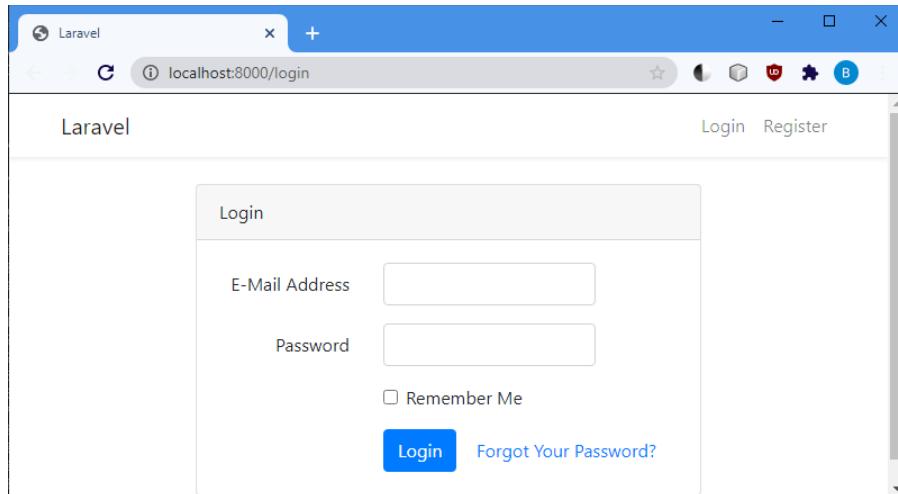
Przejdź do pliku **app.blade.php** i wykorzystaj bibliotekę Bootstrap w celu szybszego ostylowania wybranych elementów formularza logowania. W sekcji **head** dokumentu dodaj wiersze z Listingu 12.4 (konieczne do pobrania wybranych plików **css** i **js** dla Bootstrap).

#### Listing 12.4. Dodanie stylów i skryptów Bootstrap

```
<link
 rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
 integrity="sha384-gg0YR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
 crossorigin="anonymous">
<link
 rel="stylesheet"
 href="https://unpkg.com/bootstrap-table@1.15.5/dist/bootstrap-table.min.css">
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
 integrity="sha384-q8i/X+965Dz00rT7abK41JStQIAqVgRVzbzo5smXKp4YfRvH+8abTE1Pi6jizo"
 crossorigin="anonymous">
</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
 integrity="sha384-U02eT0CphqdSJQ6hJty5KVphtPhzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
 crossorigin="anonymous">
</script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
 integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFF/nJGzIxFDsf4x0xIM+B07jRM"
 crossorigin="anonymous">
</script>
```

Po uruchomieniu serwera deweloperskiego z wiersza poleceń i wpisaniu adresu strony z formularzem logowania, będzie się on prezentował jak na rysunku 12.32. Formularz

rejestracji (**register.blade.php**) także korzysta z szablonu **app.blade.php**, wobec czego automatycznie będzie wyświetlany w tym samy stylu (sprawdź).



Rys. 12.32. Widok formularza logowania po modyfikacjach

W celu dodania własnych reguł CSS, w tym samym pliku, w sekcji **head** dodaj jeszcze link:

```
<link href="{{ asset('css/custom_style.css') }}" rel="stylesheet">
```

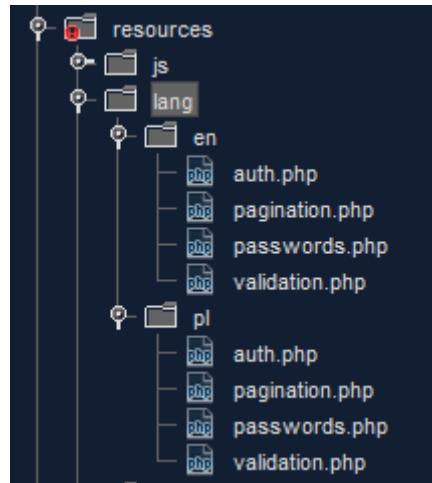
A w katalogu **public/css** utwórz plik **custom\_style.css**. W pliku tym można definiować własne style, np. zmienić kolor niebieskiego tła dla przycisku **Login** dodając do nowo utworzonego pliku regułę jak na Listingu 12.5.

#### **Listing 12.5. Stylowanie przycisku**

```
button[type="submit"]
{
 background-color: rebeccapurple;
 border: rebeccapurple;
}
```

## **2. Komunikaty o błędach validacji w j. polskim**

W celu dodania komunikatów z validatora w języku polskim, skopiuj zawartość katalogu **resources/lang/en** i umieść ją w analogicznym katalogu **resources/lang/pl** (Rys. 12.33).



Rys. 12.32. Zawartość katalogu *lang/pl*

Przejdź do pliku *resources/lang/pl/validation.php* i zmień wartości np. dla elementów tablicy o kluczu 'min' na polskie odpowiedniki (Listing 12.6).

#### Listing 12.6. Ustawienie polskich komunikatów w walidacji

```
'min' => [
 'numeric' => 'Pole :attribute musi zawierać liczbę nie mniejszą niż :min.',
 'file' => 'Plik :attribute musi zawierać minimum :min kilobajtów.',
 'string' => 'Pole :attribute musi zawierać co najmniej :min znaków.',
 'array' => 'Pole :attribute powinno zawierać minimum :min elementów.',
],
```

Pole *textarea*, w którym użytkownik wpisuje treść komentarza, posiada wartość atrybutu *name="message"*. Można przetłumaczyć także ten tytuł pola na polski odpowiednik. W tym celu w pliku *validation.php* dodaj nową wartość do tablicy asocjacyjnej *attributes* (na samym końcu pliku). Jej kluczem będzie nazwa pola, a wartością - polski odpowiednik (Rys. 12.33).

```
/*
| -----
| Custom Validation Attributes
| -----
|
| The following language lines are used to swap our attribute placeholder
| with something more reader friendly such as "E-Mail Address" instead
| of "email". This simply helps us make our message more expressive.
|
*/
'attributes' => [
 'message' => 'komentarz',
],
];
```

Rys. 12.33. Ustawienie tłumaczenia własnych atrybutów



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



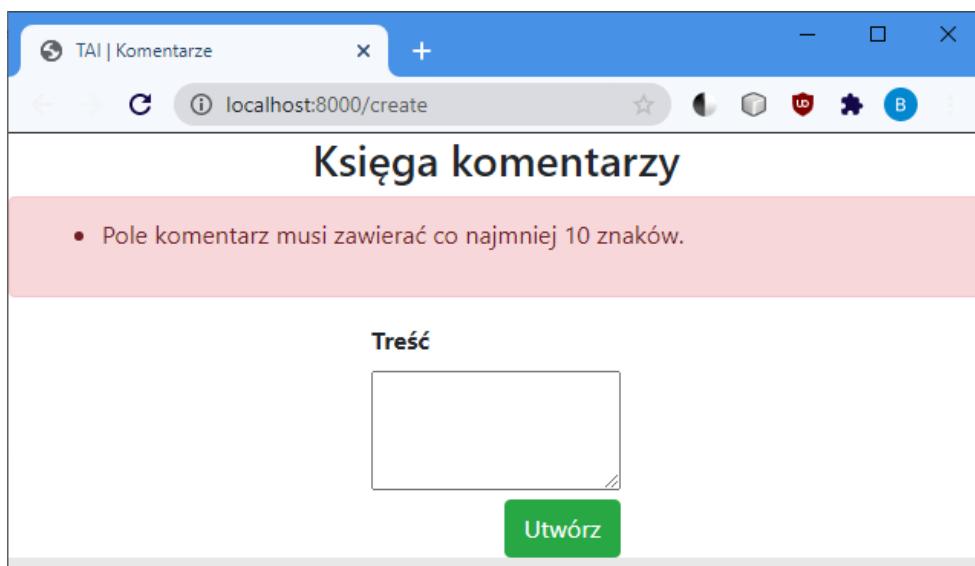
Unia Europejska  
Europejski Fundusz Społeczny

Aby komunikaty o błędach były wyświetlane w języku polskim, należy jeszcze zmodyfikować wartość parametru **locale** w pliku **config/app.php** (Rys. 12.34).

```
/*
|--
| Application Locale Configuration
|--
|
| The application locale determines the default locale that will be used
| by the translation service provider. You are free to set this value
| to any of the locales which will be supported by the application.
|
*/
'locale' => 'pl',
```

Rys. 12.34. Ustawienie w pliku konfiguracyjnym domyślnego języka polskiego

Przetestuj działanie validatora w połączeniu z modułem tłumaczeń (Rys. 12.35).



Rys. 12.35. Komunikat walidacji w języku polskim

Laravel posiada bardzo prosty, ale jednocześnie dający bardzo duże możliwości, moduł do tłumaczeń elementów strony. Warto uwagi jest fakt, że Laravel sam dba o ewentualny brak tłumaczenia w danym języku. Jeśli np. aplikacja ma możliwość tłumaczenia na kilka języków, ale nie każde tłumaczenie jest w 100% ukończone, to dzięki opcji:

```
'fallback_locale' => 'en',
```

domyślnym językiem jest język angielski. W przypadku braku tłumaczenia dla jakiegoś klucza, zostanie automatycznie zastosowany zwrot w języku angielskim.

### 3. Połaczenie ze stroną domową Laravel

W przykładowej aplikacji Laravel, domyślny widok dla strony głównej (**welcome.blade.php**) jest postaci jak na Rys. 12.36.



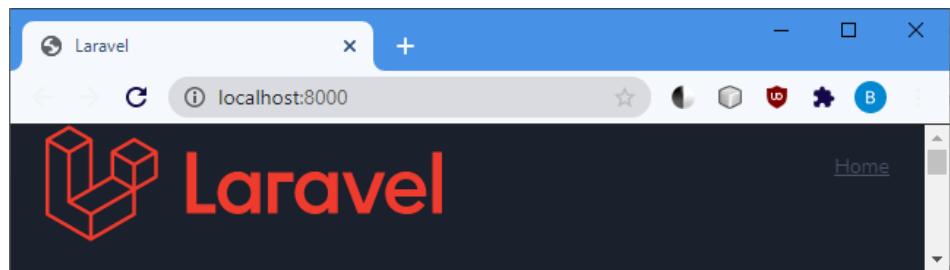
Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny



Rys. 12.36. Domyślny widok strony `welcome.blade.php`

W aplikacji tworzonej na poprzednim laboratorium były definiowane dodatkowe reguły routingu. Sprawdź, czy Twoje reguły w pliku `routes/web.php` są zgodne z tymi na Rys. 12.37.

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\CommentsController;
use App\Http\Controllers\HomeController;

/*
|--
| Web Routes
|--
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
Route::get('/', function() {
 return view('welcome');
});
Route::get('/home',[HomeController::class,'index'])->name('home');
Route::get('/comments',[CommentsController::class,'index'])->name('comments');
Route::get('/create',[CommentsController::class,'create'])->name('create');
Route::post('/create',[CommentsController::class,'store'])->name('store');
```

Rys. 12.37. Reguły routingu

Punktem wejścia do projektu jest widok `welcome.blade.php`. Ponadto plik `routes/web.php` zawiera definicje tras utworzonych na poprzednim laboratorium dla kontrolera `CommentsController`.

Jeśli reguły routingu są już takie same, uprość i zmodyfikuj stronę `welcome.blade.php` do postaci podobnej jak na rysunku 12.38.

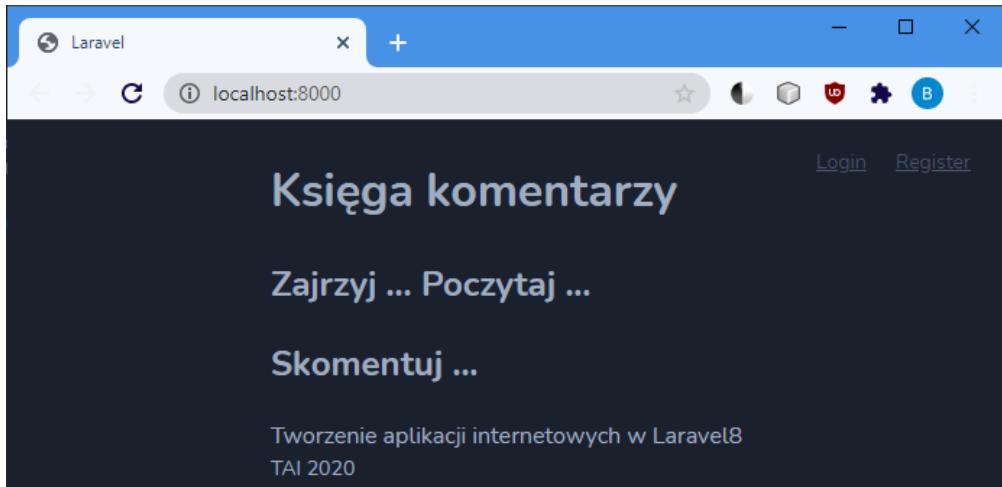


Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska





Rys. 12.38. Przykładowy wygląd strony *welcome.blade.php* po modyfikacjach

Kolejnym etapem będzie podłączenie menu pod **Księgę komentarzy**, celem łatwego powrotu do strony głównej. Wykorzystaj możliwości szablonów Blade i w folderze *resources/views/layouts* utwórz element nawigacyjny strony *navbar.blade.php* z zawartością jak na Listingu 12.7.

#### Listing 12.7. Szablon *navbar.blade.php*

```
<nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">
 <div class="container">
 Powitanie
 Komentarze
 <button class="navbar-toggler" type="button" data-toggle="collapse"
 data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
 aria-expanded="false" aria-label="{{ __('Toggle navigation') }}>

 </button>
 <div class="collapse navbar-collapse" id="navbarSupportedContent">
 <!-- Left Side Of Navbar -->
 <ul class="navbar-nav mr-auto">

 <!-- Right Side Of Navbar -->
 <ul class="navbar-nav ml-auto">

 <!-- Authentication Links -->
 @guest
 <li class="nav-item">
 {{ __('Login') }}

 @if (Route::has('register'))
 <li class="nav-item">
 {{ __('Register') }}

 @endif
 @else
 <li class="nav-item dropdown">
 <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#"
 role="button" data-toggle="dropdown" aria-haspopup="true"
 aria-expanded="false" v-pre>
 {{ Auth::user()->name }}

 @endelse

 </div>
 </div>
</nav>
```



Fundusze  
Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

```

<div class="dropdown-menu dropdown-menu-right"
 aria-labelledby="navbarDropdown">
<a class="dropdown-item" href="{{ route('logout') }}"
 onclick="event.preventDefault();
 document.getElementById('logout-form').submit();">
 {{ __('Logout') }}
<form id="logout-form" action="{{ route('logout') }}"
 method="POST" style="display: none;">
 @csrf
</form>
</div>

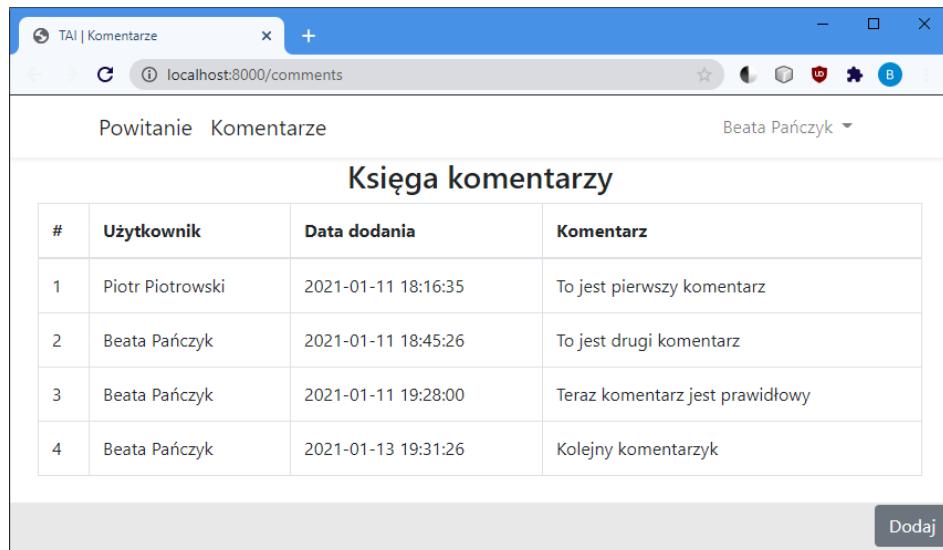
@endguest

</div>
</div>
</nav>
```

Element z nawigacją dołącz do pliku **resources/views/comments.blade.php** za pomocą **include** (zaraz po elemencie **body**):

```
@include('layouts.navbar')
```

Pozwoli to załadować zawartość menu z szablonu do strony z komentarzami (Rys. 12.39).



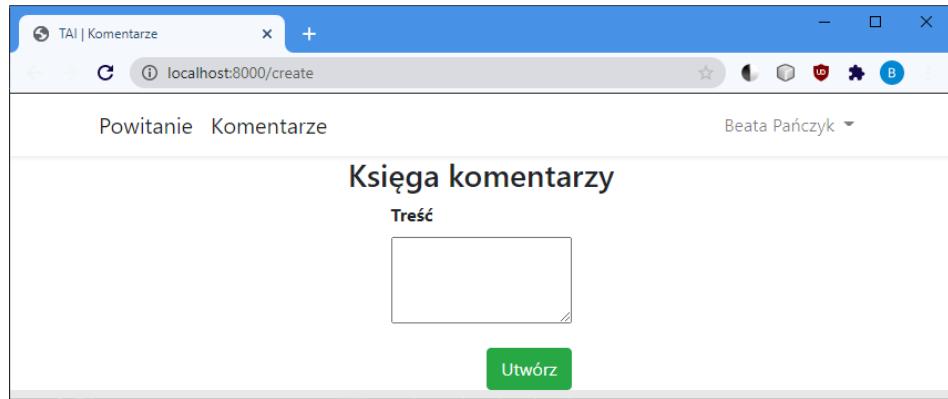
The screenshot shows a web browser window with a blue header bar. The title bar says "TAI | Komentarze". The address bar shows "localhost:8000/comments". The main content area has a header "Księga komentarzy". Below it is a table with four columns: "#", "Użytkownik", "Data dodania", and "Komentarz". The table contains four rows of data:

| # | Użytkownik       | Data dodania        | Komentarz                       |
|---|------------------|---------------------|---------------------------------|
| 1 | Piotr Piotrowski | 2021-01-11 18:16:35 | To jest pierwszy komentarz      |
| 2 | Beata Pańczyk    | 2021-01-11 18:45:26 | To jest drugi komentarz         |
| 3 | Beata Pańczyk    | 2021-01-11 19:28:00 | Teraz komentarz jest prawidłowy |
| 4 | Beata Pańczyk    | 2021-01-13 19:31:26 | Kolejny komentarzyk             |

A "Dodaj" button is visible at the bottom right of the table.

Rys. 12.39. Widok strony komentarzy z menu

W celu zachowania spójności na stronach aplikacji, element nawigacyjny należy w ten sam sposób dodać do **commentsForm.blade.php** (Rys. 12.40).



Rys. 12.40. Widok formularza dodawania komentarzy z menu

#### 4. Usuwanie komentarzy z bazy danych

Ostatnim etapem prac nad projektem jest dodanie możliwości kasowania swoich komentarzy przez autorów wpisów. Dodajmy najpierw kolejną regułę routingu do pliku **routes/web.php**:

```
Route::get('/delete/{id}',
 [CommentsController::class, 'destroy'])->name('delete');
```

Reguła ta definiuje obsługę URL postaci **delete/id** przez metodę **delete(\$id)** (z parametrem **\$id**) kontrolera **CommentsController**. Przy usuwaniu komentarza ważne będzie sprawdzenie, czy użytkownik usuwa swój komentarz (czy ma do tego uprawnienia). W pliku kontrolera **app/Http/Controllers/CommentsController.php**, uzupełnij istniejącą już tam metodę **destroy** (Listing 12.8).

##### Listing 12.8. Dodatkowa metoda delete w CommentsController.php

```
public function destroy($id)
{
 ...//Znajdź komentarz o danych id:
 $comment = Comment::find($id);
 //Sprawdź czy użytkownik jest autorem komentarza:
 if(\Auth::user()->id != $comment->user_id)
 {
 return back();
 }
 if($comment->delete()){
 return redirect()->route('comments');
 }
 else return back();
}
```

Aby użytkownik mógł wykonać akcję usunięcia tylko **swojego** wpisu, dodaj jeszcze przycisk kasowania komentarza, który powinien wyświetlać się tylko gdy dany komentarz może być skasowany. W tym celu, w widoku **resources/views/comments.blade.php**, w pętli wyświetlającej tabelę komentarzy, dodaj, w ostatniej komórce tabeli, instrukcję **@if** jak na Listingu 12.9.



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczpospolita  
Polska



Unia Europejska  
Europejski Fundusz Społeczny

**Listing 12.9. Dodatkowy blok @if w pętli widoku comments.blade.php**

```
<tbody>
 @foreach($comments as $comment)
 <tr>
 <td>{{$comment->id}}</td>
 <td>{{$comment->user->name}}</td>
 <td>{{$comment->created_at}}</td>
 <td>{{$comment->message}}</td>

 @if($comment->user_id == \Auth::user()->id)
 id) }}"
 class="btn btn-danger btn-xs"
 onclick="return confirm('Jesteś pewien?')"
 title="Skasuj"> Usuń

 @endif
 </td>
 </tr>
 @endforeach
</tbody>
```

Wygląd strony po modyfikacjach przedstawia rysunek 12.41. Przetestuj działanie przycisku **Usuń**.

| # | Użytkownik       | Data dodania        | Komentarz                                      |
|---|------------------|---------------------|------------------------------------------------|
| 1 | Piotr Piotrowski | 2021-01-11 18:16:35 | To jest pierwszy komentarz                     |
| 3 | Beata Pańczyk    | 2021-01-11 19:28:00 | Teraz komentarz jest prawidłowy<br><b>Usuń</b> |
| 4 | Beata Pańczyk    | 2021-01-13 19:31:26 | Kolejny komentarzyk<br><b>Usuń</b>             |

Rys. 12.41. Widok strony komentarza z przyciskiem do usuwania

## 5. Edycja komentarzy

Zacznij od dodania kolejnych reguł routingu koniecznych do obsługi modyfikacji danych, jak na Listingu 12.10.

**Listing 12.10. Dodatkowe reguły routingu**

```
Route::get('/edit/{id}', [CommentsController::class, 'edit'])->name('edit');
Route::put('/update/{id}', [CommentsController::class, 'update'])->name('update');
```

W nowych regułach dodano odniesienie do nowej (ale już istniejącej w kontrolerze) metody **edit** kontrolera **CommentsController**, odpowiedzialnej za załadowanie formularza do edycji komentarza. Kolejna metoda **update** tego kontrolera, odpowiadać będzie za przetworzenie



Fundusze Europejskie  
Wiedza Edukacja Rozwój



Rzeczypospolita  
Polska



danych zmodyfikowanego komentarza, wysłanych z formularza edycji. W kontrolerze **CommentsController** uzupełnij te istniejące już, ale jeszcze puste metody zgodnie z kodem na Listingu 12.11.

**Listing 12.11. Uzupełnione funkcje edit i update**

```
public function edit($id) {
 $comment = Comment::find($id);
 //Sprawdzenie czy użytkownik jest autorem komentarza
 if (\Auth::user()->id != $comment->user_id) {
 return back()->with(['success' => false, 'message_type' => 'danger',
 'message' => 'Nie posiadasz uprawnień do przeprowadzenia tej operacji.']);
 }
 return view('commentsEditForm', ['comment'=>$comment]);
}
public function update(Request $request, $id)
{
 $comment = Comment::find($id);
 //Sprawdzenie czy użytkownik jest autorem komentarza
 if (\Auth::user()->id != $comment->user_id)
 {
 return back()->with(['success' => false, 'message_type' => 'danger',
 'message' => 'Nie posiadasz uprawnień do przeprowadzenia tej operacji.']);
 }
 $comment->message = $request->message;
 if ($comment->save()) {
 return redirect()->route('comments');
 }
 return "Wystąpił błąd.";
}
```

Widok listy komentarzy można teraz uzupełnić dodatkowym przyciskiem **Edytuj** - w pliku **comments.blade.php** w bloku **@if**, ale przed przyciskiem **Usuń**, dodaj kod:

```
<a href="{{ route('edit', $comment) }}" class="btn btn-success btn-xs"
 title="Edytuj"> Edytuj
```

Po uruchomieniu strony z komentarzami, dla zalogowanego użytkownika, otrzymamy teraz widok jak na Rys. 12.42.

Potrzebujemy jeszcze tylko formularza do modyfikacji wskazanego komentarza - pliku **commentsEditForm.blade.php** (Listing 12.12). Zauważ, że formularz jest analogiczny do formularza dodawania komentarza.

Są tam 3 różnice (oznaczone kolorem czerwonym na Listingu 12.12):

- wartość atrybutu **action** formularza (dane komentarza przesłane metodą POST mają teraz trafić do metody **update**, zgodnie z regułą routingu o nazwie ‘**update**’),
- dodano treść wskazanego do edycji komentarza (pole **textarea**),
- dodano ukryte pole wskazujące metodę PUT protokołu HTTP.

**UWAGA!**

POST jest używany do tworzenia zasobu, PUT do tworzenia lub aktualizowania zasobu.

Ponieważ formularze HTML obsługują **tylko** metody POST i GET, metody PUT i DELETE można symulować przez dodanie do formularza ukrytego pola z atrybutem **name="method"**.

| # | Użytkownik       | Data dodania        | Komentarz                                                                    |
|---|------------------|---------------------|------------------------------------------------------------------------------|
| 1 | Piotr Piotrowski | 2021-01-11 18:16:35 | To jest pierwszy komentarz                                                   |
| 4 | Beata Pańczyk    | 2021-01-13 19:31:26 | Kolejny komentarzyk                                                          |
| 5 | Beata Pańczyk    | 2021-01-14 19:07:10 | Do pracy z bazami danych w Laravel wygodnie jest korzystać z ORM - Eloquent. |
| 6 | Anna             | 2021-01-14 19:20:18 | To jest bardzo fajny framework.                                              |

Rys. 12.42. Widok strony komentarza z przyciskami *Edytuj* i *Usuń*

**Listing 12.12. Formularz do edycji komentarza – fragment pliku *commentsEditForm.blade.php***

```
<form role="form" id="comment-form" method="post"
 action="{{ route('update', $comment) }}"
 {{ csrf_field() }}
 <input name="_method" type="hidden" value="PUT">
 <div class="box">
 <div class="box-body">
 <div class="form-group{{ $errors->has('message') ? ' has-error' : '' }}>
 <label>Treść</label>

 <textarea name="message" id="message" cols="30" rows="10" required>
 {{$comment->message}}
 </textarea>
 </div>
 </div>
 </div>
 <div class="box-footer">
 <button type="submit" class="btn btn-success">Zapisz</button>
 </div>
 </form>
```

Sprawdź efekt działania przycisku ***Edytuj***.

Przy modyfikacji zasobu można też klasycznie zastosować metodę POST – zmień odpowiednio regułę w pliku *web.php*, usuń ukryte pole z formularza i przetestuj ponownie działanie edycji.

Sprawdź działanie całej aplikacji, ewentualnie dokonaj (według uznania) modyfikacji w nawigacji pomiędzy stronami.



## **LABORATORIUM 13. PREZENTACJA PROJEKTÓW ZALICZENIOWYCH**

### **Cel laboratorium:**

Celem zajęć jest prezentacja przez studentów projektów aplikacji internetowych wykonanych zgodnie z zadaną specyfikacją na zaliczenie przedmiotu.

### **Zakres tematyczny zajęć:**

Prezentacja projektów przez studentów.

Dyskusja na temat wykorzystanych rozwiązań.

Ocena projektów.



**Fundusze  
Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita  
Polska**

**Unia Europejska**  
Europejski Fundusz Społeczny





Materiały zostały opracowane w ramach projektu  
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,  
umowa nr **POWR.03.05.00-00-Z060/18-00**  
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020  
współfinansowanego ze środków Europejskiego Funduszu Społecznego



**Fundusze Europejskie**  
Wiedza Edukacja Rozwój



**Rzeczpospolita Polska**



**Unia Europejska**  
Europejski Fundusz Społeczny