

Powtórka: strona z formularzem i skryptem

Czas: 2 × 45 min

praca indywidualna (zadanie diagnostyczne)

Cel: Sprawdzenie, co pamiętasz z roku 1: zbudujesz formularz rezerwacji, napiszesz skrypt walidujący dane i liczący koszt, a na końcu ocenisz swoje umiejętności i zaplanujesz powtórkę.

Po zajęciach potrafisz:

- tworzy formularz HTML z etykietami i polami różnych typów
- sprawdza dane z formularza w JavaScript i pokazuje komunikaty przy polach
- oblicza wynik według reguły z polecenia i wyświetla go na stronie
- dodaje elementy do tabeli przez DOM bez używania innerHTML dla danych użytkownika
- ocenia własne umiejętności i wskazuje tematy do powtórki

Sprzęt i materiały

- komputer z Windows 11
- folder `r2_01_start` z Pliki_do_zadan (`index.html`, `styl.css`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (podgląd przez `http://127.0.0.1:5500`)
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przejrzyj notatki z roku 1: formularze (1.05), zdarzenia i DOM (1.17).

Bezpieczeństwo

- Tekst wpisany przez użytkownika wstawiaj na stronę przez `textContent`, nie przez `innerHTML`.
- Pracuj samodzielnie — wynik ma pokazać Twój rzeczywisty poziom, a nie poziom kolegi lub generatora kodu.

Przebieg zadania

1 Przygotowanie projektu

Wypożyczalnia kajaków potrzebuje strony, na której klient sprawdzi koszt spływu i zarezerwuje kajaki. Układ strony i style są gotowe — Ty dopisujesz formularz i skrypt.

1. Utwórz folder `nazwisko_imie_z01`, skopiuj do niego pliki z `r2_01_start` i otwórz folder w VS Code. Uruchom podgląd przez Live Server.
2. W stopce wpisz swoje imię, nazwisko i klasę.

2 Formularz (HTML)

Każde pole ma etykietę `label` powiązaną atrybutem `for`, a pod polem akapit na komunikat, np. `<p class="blad" id="blad-email"></p>`.

1. W sekcji `#formularz` wstaw formularz `id="rezerwacja"` z atrybutem `novalidate` (walidację zrobisz w JS). Pola: imię i nazwisko (`id="osoba"`), e-mail (`type="email"`, `id="email"`), data spływu (`type="date"`, `id="data"`), liczba kajaków (`type="number"`, `id="kajaki"`, od 1 do 6, domyślnie 1).
2. Dodaj listę `select id="trasa"` z pustą opcją „— wybierz trasę —” i trzema trasami, których `value` to cena za kajak: Krótka 12 km — 60 zł, Średnia 20 km — 85 zł, Długa 32 km — 120 zł.
3. Dodaj dwa pola wyboru: transport powrotny (`id="transport"`, +15 zł za kajak) i akceptacja regulaminu (`id="regulamin"`, klasa etykiety `check`), a na końcu przyciski `submit` „Oblicz i zarezerwuj” oraz `reset` „Wyczyść”.

3 Walidacja (JavaScript)

Utwórz plik `skrypt.js` i podłącz go na końcu `body`. Klasa `pole-blad` z CSS podświetla błędne pole na czerwono.

1. Napisz funkcję `pokazBlad(id, tekst)`, która wpisuje tekst do `#blad-<id>` i dodaje polu klasę `pole-blad` (pusty tekst — usuwa komunikat i klasę). Funkcja zwraca `true`, gdy błędu nie ma.
2. Napisz funkcję `sprawdz()`: imię i nazwisko min. 3 znaki (po `trim()`), e-mail zgodny z wyrażeniem regularnym (znaki, @, domena z kropką), data wybrana i nie wcześniejsza niż dzisiejsza, wybrana trasa, liczba kajaków całkowita 1–6, zaznaczony regulamin. Sprawdź **wszystkie** pola naraz, żeby użytkownik zobaczył wszystkie błędy.

4 Obliczenia i wynik

Reguły cennika: koszt = cena trasy × liczba kajaków; transport dodaje 15 zł za każdy kajak; przy 4 i więcej kajakach rabat 10% od całości. Kwoty pokazuj z dwoma miejscami po przecinku i „zł”, np. `360,00 zł`.

1. Obsłuż zdarzenie `submit`: zablokuj wysłanie (`preventDefault`), wywołaj `sprawdz()`, a przy poprawnych danych oblicz koszt.
2. W `#wynik` pokaż podsumowanie: imię i nazwisko, nazwę trasy, liczbę kajaków, ewentualny rabat i kwotę do zapłaty (usuń klasę `ukryty`). Tekst wstawiaj przez `textContent`.
3. Dodaj wiersz do tabeli „Twoje rezerwacje” (`createElement`, `appendChild`) i aktualizuj sumę w `#razem`.
4. Obsłuż zdarzenie `reset`: usuń komunikaty błędów i ukryj podsumowanie.
5. Przetestuj i zapisz w sprawozdaniu wyniki: T1 — wysłanie pustego formularza; T2 — trasa Średnia, 4 kajaki, z transportem; T3 — trasa Krótka, 2 kajaki, bez transportu (suma w tabeli po T2 i T3); T4 — data wczorajsza.

5 Samoocena i mapa zaległości

Ta część jest równie ważna jak kod — na jej podstawie zaplanujemy powtórki.

1. W tabeli samooceny w sprawozdaniu oceń każdą umiejętność w skali 0–2 (0 — nie umiem, 1 — z pomocą, 2 — samodzielnie) i zaznacz, czy w tym zadaniu korzystałeś z notatek.
2. Wypisz 3 tematy z roku 1, które musisz powtórzyć, i przy każdym wskaż materiał (numer wykładu lub zadania roku 1). Zrób zrzuty ekranu T1 i T2 oraz spakuj folder do ZIP.

W sprawozdaniu umieść

- ☐ zrzuty ekranu testów T1 (komunikaty błędów) i T2 (podsumowanie z rabatem)
- ☐ tabela testów T1–T4 z oczekiwanym i otrzymanym wynikiem
- ☐ fragmenty kodu: funkcja `sprawdz()` i obliczanie kosztu

- ☐ tabela samooceny i lista 3 tematów do powtórki
- ☐ spakowany folder `nazwisko_imie_z01`

Pytania kontrolne

1. Po co formularzowi atrybut `novalidate`, skoro pola mają typy `email` i `number`?
2. Dlaczego dane od użytkownika lepiej wstawiać przez `textContent` niż `innerHTML`?
3. Co by się stało bez `preventDefault()` w obsłudze zdarzenia `submit`?
4. Dlaczego walidacja w JavaScript nie wystarczy, gdy rezerwacja trafia na serwer?

Kryteria oceny

Kryterium	Punkty
Formularz: wszystkie pola z etykietami, poprawne typy, select z trzema trasami, przyciski submit i reset	4
Walidacja: 6 warunków, komunikaty przy polach, podświetlenie błędnych pól, wszystkie błędy naraz	5
Obliczenia zgodne z cennikiem (T2 = 360,00 zł, suma po T3 = 480,00 zł), format kwot	4
Tabela rezerwacji przez DOM, suma, obsługa reset, <code>textContent</code> dla danych	3
Samoocena, lista tematów do powtórki, zrzuty i odpowiedzi na pytania	4
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Nowoczesny JavaScript w praktyce

Czas: 2 × 45 min

praca indywidualna

Cel: Przepisanie działającego, ale przestarzałego skryptu sklepu na współczesny JavaScript: moduły ES, szablony tekstowe, funkcje strzałkowe, destrukuryzację, spread, operatory `?.` i `??` oraz klasę.

Po zajęciach potrafisz:

- zastępuje `var` deklaracjami `let` / `const` i konkatenację szablony tekstowymi
- dzieli kod na moduły ES z `export` / `import` i uruchamia je przez `type="module"`
- stosuje destrukuryzację, spread, parametry domyślne oraz operatory `?.` i `??`
- pisze klasę z polem prywatnym, metodami i getterami
- porównuje działanie starej i nowej wersji tymi samymi testami

Sprzęt i materiały

- komputer z Windows 11
- folder `r2_02_start` z Pliki_do_zadan (`index.html`, `styl.css`, `stary.js`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (podgląd przez `http://127.0.0.1:5500`)
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)
- **Git** — opcjonalnie: commit przed i po refaktoryzacji
git-scm.com/downloads

Przygotowanie przed zajęciami

- Przeglądaj wykład 2.01 (ES6+, moduły, klasy).

Bezpieczeństwo

- Nazwy produktów i opisy wstawiaj przez `textContent`; w szablonie HTML umieszczaj tylko liczby i stałe teksty.
- Moduły ES nie działają z adresu `file://` — otwieraj stronę wyłącznie przez Live Server (`http://127.0.0.1:5500`).

Przebieg zadania

1 Stara wersja i testy odniesienia

Najpierw sprawdź, jak działa stary kod — po przepisaniu wyniki muszą być takie same.

1. Skopiuj `r2_02_start` do folderu `nazwisko_imie_z02`, otwórz go w VS Code i uruchom Live Server. Jeśli używasz Gita, zrób commit „stara wersja”.
2. Wykonaj test odniesienia: dodaj do koszyka 2 × „Plecak Fjord 30 l” i 1 × „Namiot Polaris 2-os.”. Zapisz sumę. Wybierz kategorię „akcesoria” i zapisz liczbę kart.
3. Przeczytaj `stary.js` i wypisz w sprawozdaniu co najmniej 6 miejsc do unowocześnienia (np. `var`, pętla szukająca produktu, `rabat === undefined`).

2 Moduły: dane i narzędzia

Docelowa struktura: `dane.js`, `narzedzia.js`, `koszyk.js`, `main.js`. W `index.html` zamień `<script src="stary.js">` na `<script type="module" src="main.js"></script>`. W ścieżkach importu pisz `./` i rozszerzenie `.js`.

```
// narzedzia.js – przykład eksportu
export const zl = (kwota) => `${kwota.toFixed(2).replace(".", ",")} zł`;

// main.js – przykład importu
import { zl } from "./narzedzia.js";
```

- W `dane.js` wyeksportuj tablicę `produkty` (`export const`) oraz obiekt `nowosc`: Mata Puch, akcesoria, 119 zł, stan 5, id 9.
- W `narzedzia.js` wyeksportuj funkcje strzałkowe `zl(kwota)` i `cenaPoRabacie({ cena, rabat })` — z destrukuryzacją parametru i operatorem `??` zamiast warunku `=== undefined`.
- Dodaj eksport domyślny `kategorie(lista)`, który zwraca kategorie bez powtórzeń przy użyciu `Set` i spreadu (`[...new Set(...)]`) zamiast pętli z `indexOf`.

3 Klasa Koszyk

Koszyk przechowuje pozycje w prywatnym polu `#pozycje` typu `Map` (klucz — id produktu, wartość — `{ produkt, ilosc }`).

- W `koszyk.js` napisz i wyeksportuj klasę `Koszyk` z metodami `dodaj(produkt, ilosc = 1)`, `usun(id)`, `wyczysc()`. W `dodaj` odczytaj obecną ilość wyrażeniem `this.#pozycje.get(id)?.ilosc ?? 0`.
- Dodaj gettery: `pozycje` (tablica wartości z `Map`), `liczba` (suma sztuk) i `suma` (kwota po rabatach) — dwa ostatnie przez `reduce`.
- Gdy strona już działa (krok 4), dopisz tymczasowo na końcu `main.js` linię `console.log(koszyk.#pozycje);`, odśwież stronę, zanotuj błąd z konsoli F12 i usuń tę linię.

4 main.js — interfejs

Główny moduł importuje pozostałe i obsługuje stronę. Nie zmieniaj tablicy z `dane.js` — twórz kopie.

- Utwórz tablicę `produkty` jako `[...bazowe, { ...nowosc, rabat: 5 }]` (spread tablicy i obiektu). Wyświetlaj karty funkcją `karta({ id, nazwa, opis, stan, rabat, ...reszta })` z szablonem tekstowym; opis wstaw przez `textContent` z `opis ?? ""`.
- Obsłuż filtr kategorii i nową listę „Sortuj” (cena rosnąco, nazwa A–Z): sortuj kopię `[...lista].sort(...)`, nazwy porównuj `localeCompare(..., "pl")`.
- Kliknięcia „Do koszyka” obsłuż jednym nasłuchem na `#produkty` (delegowanie zdarzeń, `e.target.dataset.id`). W koszyku pokaż przycisk `×` usuwający pozycję oraz działający przycisk „Wyczyść koszyk”; pod sumą wyświetl liczbę sztuk.
- Powtórz test odniesienia z kroku 1 — suma musi być identyczna. Sprawdź, że „Mata Puch” ma cenę po rabacie 5%. Zrób zrzut ekranu i (opcjonalnie) commit „wersja ES6+”.

W sprawozdaniu umieść

- ☐ wynik testu odniesienia dla starej i nowej wersji (suma koszyka, liczba akcesoriów)
- ☐ tabela „stary kod → nowy kod” z co najmniej 6 zmianami
- ☐ kod klasy `Koszyk` i komunikat przy próbie odczytu pola prywatnego
- ☐ zrzut ekranu nowej wersji z koszykiem i sortowaniem
- ☐ spakowany folder `nazwisko_imie_z02` (lub link do repozytorium)

Pytania kontrolne

1. Czym różni się `??` od `||`? Podaj przykład z produktem, który ma `rabat: 0`.
2. Dlaczego strona z modułami nie działa po otwarciu pliku dwuklikiem (adres `file://`)?
3. Czym różni się eksport nazwany od domyślnego i jak wygląda import każdego z nich?
4. Dlaczego przed `sort()` robimy kopię tablicy `[...lista]`?

Kryteria oceny

Kryterium	Punkty
Analiza starego kodu i test odniesienia (min. 6 miejsc do zmiany)	3
Moduły dane.js i narzedzia.js: eksporty nazwane i domyślny, Set, ??, destrukuryzacja	4
Klasa Koszyk: pole prywatne, Map, 3 metody, 3 gettery, poprawna suma	5
main.js: szablony, spread, filtr i sortowanie kopii, delegowanie zdarzeń, usuwanie z koszyka	5
Zgodność wyników starej i nowej wersji, zrzuty, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Formularz z walidacją i localStorage

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie formularza kontaktowego z walidacją na żywo (atrybuty HTML, Constraint Validation API, wyrażenia regularne) oraz zapisywanie szkicu i listy notatek w localStorage tak, aby przetrwały odświeżenie strony.

Po zajęciach potrafisz:

- dobiera atrybuty walidacji HTML (`required`, `minlength`, `pattern`, typy pól)
- odczytuje stan pola przez `validity` i wyświetla własne komunikaty, także z `setCustomValidity`
- zapisuje i odczytuje dane w localStorage z użyciem `JSON.stringify` i `JSON.parse`
- tworzy listę zadań zachowywaną po odświeżeniu strony
- wyjaśnia różnicę między localStorage i sessionStorage oraz ograniczenia pamięci przeglądarki

Sprzęt i materiały

- komputer z Windows 11
- folder `r2_03_start` z Pliki_do_zadan (`index.html`, `styl.css`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (podgląd przez `http://127.0.0.1:5500`)
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przejrzyj wykład 2.02 (Constraint Validation API, localStorage).

Bezpieczeństwo

- localStorage może odczytać każdy skrypt działający na stronie i każdy, kto usiądzie przy komputerze — nie zapisuj w nim haseł, numerów PESEL ani danych kart.
- Po zajęciach wyczyść dane strony: F12 → Aplikacja (Application) → Pamięć lokalna → Wyczyść, szczególnie na wspólnym komputerze w pracowni.

Przebieg zadania

1 Atrybuty walidacji w HTML

Pracownia ceramiki chce formularza kontaktowego i podręcznej listy notatek. Skopiuj `r2_03_start` do folderu `nazwisko_imie_z03` i uruchom Live Server.

1. Dodaj atrybuty: imię i nazwisko — wymagane, 3–60 znaków; e-mail — `type="email"`, wymagany; telefon — `type="tel"`, nieobowiązkowy, wzorzec 9 cyfr z opcjonalnymi spacjami po 3 cyfrach; kod pocztowy — wymagany, wzorzec `00-000`; temat i zgoda — wymagane; wiadomość — 20–500 znaków.
2. Usuń na chwilę `novalidate` z formularza, wyślij go pusty i zrób zrzut dymka przeglądarki. Przywróć `novalidate` — dalej komunikaty pokazuje Twój skrypt.

2 Walidacja na żywo (formularz.js)

Każde pole ma pod sobą akapit `#k-<id pola>`. Klasy `ok` i `zle` kolorują ramkę pola.

```
function tekstBledu(pole) {
  const v = pole.validity;
  if (v.valueMissing) return "To pole jest wymagane.";
  // dopisz: typeMismatch, patternMismatch (osobno kod i telefon),
  // tooShort z liczbą znaków, customError
  return "";
}
```

1. Dokończ funkcję `tekstBledu(pole)`. Dla zgody wyświetl „Zaznacz zgodę.”, dla zbyt krótkiej wiadomości np. „Za krótko: minimum 20 znaków (jest 6).”.
2. Napisz `sprawdzPole(pole)`, która wpisuje komunikat i przełącza klasy `ok` / `zle` (`classList.toggle`). Wywołuj ją przy zdarzeniach `input` i `blur` każdego pola.
3. Dodaj regułę własną przez `setCustomValidity`: imię i nazwisko musi składać się z co najmniej dwóch słów (podziel tekst wyrażeniem `/\s+/`). Pusty tekst w `setCustomValidity("")` usuwa błąd.
4. Pod wiadomością wyświetlaj licznik znaków „N / 500”. Przycisk „Wyślij” ma być nieaktywny, dopóki `form.checkValidity()` zwraca `false`.

3 Szkic w localStorage

Jeśli użytkownik zamknie kartę w połowie pisania, po powrocie formularz ma się wypełnić.

1. Przy każdym `input` zapisuj obiekt z wartościami wszystkich pól i czasem zapisu pod kluczem `glina-szkic` (`JSON.stringify`). Dla pola wyboru zapisuj `checked`.
2. Po załadowaniu strony odczytaj szkic (`JSON.parse`), wypełnij pola, sprawdź je i pokaż w `#info-szkic` napis „Przywrócono szkic z godz. GG:MM.”.
3. Po poprawnym wysłaniu usuń szkic (`removeItem`), wyczyść formularz i pokaż podziękowanie z liczbą wysłanych wiadomości w tej karcie, przechowywaną w **sessionStorage**.

4 Lista notatek (notatki.js)

Notatki to tablica obiektów `{ id, tekst, zrobione }` zapisywana w całości pod kluczem `glina-notatki`.

1. Przy starcie wczytaj tablicę: `JSON.parse(localStorage.getItem("glina-notatki") ?? "[]")`. Napisz funkcję `rysuj()`, która tworzy elementy `li` (pole wyboru, tekst przez `textContent`, przycisk „Usuń”) i pokazuje statystykę „Notatek: N, zrobionych: M”.
2. Obsłuż dodawanie (pusta notatka niedozwolona), oznaczanie jako zrobione (przekreślenie — klasa `zrobione`), usuwanie jednej notatki i przycisk „Usuń zrobione”. Po każdej zmianie zapisz tablicę i wywołaj `rysuj()`.
3. Test: dodaj 3 notatki, jedną oznacz, odśwież stronę (F5), zamknij i otwórz kartę. Obejrzyj klucze w F12 → Aplikacja → Pamięć lokalna i Pamięć sesji. Zrób zrzuty.

W sprawozdaniu umieść

- ☐ zrzut dymka walidacji przeglądarki (bez `novalidate`) i zrzut własnych komunikatów
- ☐ tabela testów pól z wpisanymi wartościami i komunikatami
- ☐ zrzut F12 → Aplikacja z kluczami `glina-szkic` i `glina-notatki` oraz zawartością `sessionStorage`
- ☐ kod funkcji `tekstBledu`, `zapiszSzkic` i `rysuj`
- ☐ spakowany folder `nazwisko_imie_z03`

Pytania kontrolne

1. Czym różni się localStorage od sessionStorage? Co stało się z licznikiem wysłanych wiadomości po zamknięciu karty?
2. Dlaczego obiekt trzeba zamienić na tekst `JSON.stringify` przed zapisem do localStorage? Co zapisze się bez tego?
3. Jakich danych nie należy przechowywać w localStorage i dlaczego?
4. Dlaczego walidacja w przeglądarce nie zwalnia serwera z ponownego sprawdzenia danych?

Kryteria oceny

Kryterium	Punkty
Atrybuty walidacji HTML zgodne z wymaganiami (7 pól, 2 wzorce)	3
Komunikaty z validity dla 5 rodzajów błędów, klasy ok/zle, reguła setCustomValidity	5
Licznik znaków i przycisk aktywny tylko dla poprawnego formularza	2
Szkic: zapis, przywrócenie z godziną, usunięcie po wysłaniu; licznik w sessionStorage	4
Notatki: dodawanie, oznaczanie, usuwanie, „Usuń zrobione”, zachowanie po odświeżeniu	4
Zrzuty z F12 i odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Fetch i publiczne API — kursy walut NBP

Czas: 2 × 45 min

praca indywidualna

Cel: Napisanie aplikacji, która pobiera aktualne kursy walut z publicznego API NBP funkcją `fetch` z `async/await`, pokazuje stan ładowania i błędy, wyświetla dane w tabeli i na wykresie.

Po zajęciach potrafisz:

- pobiera dane JSON funkcją `fetch` i obsługuje je przez `async/await`
- sprawdza `response.ok` i obsługuje błędy w `try/catch/finally`
- pokazuje użytkownikowi stan „Ładowanie...” i czytelny komunikat błędu
- tworzy tabelę z filtrowaniem i wykres liniowy z danych z API
- analizuje żądania w zakładce Sieć narzędzi F12

Sprzęt i materiały

- komputer z Windows 11 i dostępem do internetu
- folder `r2_04_start` z Pliki_do_zadan (`index.html`, `styl.css`, folder `dane`)

Oprogramowanie

- **Visual Studio Code** — rozszerzenie Live Server (podgląd przez `http://127.0.0.1:5500`)
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)
- **API NBP** — dokumentacja; dane bez klucza i bez rejestracji
api.nbp.pl
- **Chart.js 4** — dołączany z CDN:
<https://cdn.jsdelivr.net/npm/chart.js@4.4.1/dist/chart.umd.min.js>
chartjs.org/docs/latest

Przygotowanie przed zajęciami

- Przeglądnij wykład 2.03 (Promise, `async/await`, `fetch`, CORS).

Bezpieczeństwo

- Nie wysyłaj do API zapytań w pętli ani co sekundę — pobieraj dane po załadowaniu strony i po kliknięciu „Odśwież”.
- Dane z API wstawiaj przez `textContent`; nie ufaj, że zewnętrzny serwer zawsze zwróci poprawny JSON.

Przebieg zadania

1 Poznaj API

Skopiuj `r2_04_start` do folderu `nazwisko_imie_z04` i uruchom Live Server. Adresy, których użyjesz:

```
https://api.nbp.pl/api/exchangerates/tables/A/?format=json
https://api.nbp.pl/api/exchangerates/rates/A/EUR/last/10/?format=json
```

1. Otwórz oba adresy w przeglądarce. Zapisz w sprawozdaniu: czy odpowiedź jest tablicą czy obiektem, gdzie jest data tabeli i jak nazywa się pole z kursem średnim.
2. Otwórz adres z kodem nieistniejącej waluty, np. `XYZ`, i zapisz, co zwraca serwer (F12 → Sieć: kod odpowiedzi).
3. Porównaj odpowiedź z plikami `dane/tabela_a.json` i `dane/kurs_EUR.json` — mają ten sam format (to dane przykładowe do pracy bez internetu).

2 Pobieranie tabeli kursów

Utwórz `app.js` i podłącz go na końcu `body`.

```
async function pobierzJSON(url) {
  const odp = await fetch(url);
  // jeśli !odp.ok – rzuć błąd z kodem odp.status
  return odp.json();
}
```

1. Dokończ funkcję `pobierzJSON(url)`. Napisz funkcję `adres(rodzaj, kod)`, która zwraca adres API albo — gdy zaznaczono „tryb offline” — ścieżkę do pliku w folderze `dane`.
2. Napisz `async function wczytajTabele()`: pokaż w `#stan` napis „Ładowanie...” (klasa `ładowanie`) i zablokuj przycisk „Odśwież”; pobierz tabelę A (destrukuryzacja `const [tabela] = await ...`), wpisz numer i datę tabeli do `#data-tabeli`; w `catch` pokaż komunikat z klasą `bład`; w `finally` odblokuj przycisk.
3. Wyświetl kursy w tabeli: waluta, kod, kurs z 4 miejscami po przecinku (przecinek dziesiętny). Dodaj filtrowanie na żywo polem `#szukaj` (nazwa lub kod, bez względu na wielkość liter).
4. Obsłuż przycisk „Odśwież” i pole „tryb offline” (zmiana stanu — ponowne wczytanie).

3 Wykres i przelicznik

Lista `#waluta` zawiera kody z tabeli. Po wybraniu waluty pobierz jej 10 ostatnich notowań.

1. Wypełnij listę `#waluta` (opcje „EUR — euro”), domyślnie wybierz EUR.
2. Dołącz Chart.js z CDN w `head` i napisz `wczytajWykres()`: pobierz notowania, utwórz wykres liniowy (etykiety — daty, wartości — `mid`). Przed narysowaniem nowego wykresu usuń stary metodą `destroy()`.
3. Gdy waluty nie ma w trybie offline albo serwer zwróci błąd, pokaż w `#stan-wykres` komunikat, np. „Brak notowań GBP: serwer odpowiedział kodem 404”.
4. Dodaj przelicznik: kwota w zł podzielona przez kurs wybranej waluty, wynik z 2 miejscami po przecinku i kodem waluty.

4 Testy i analiza w F12

1. W F12 → Sieć znajdź żądanie tabeli A: zapisz kod odpowiedzi, typ treści (Content-Type), rozmiar i czas. Sprawdź nagłówki odpowiedzi `Access-Control-Allow-Origin`.
2. Ustaw w zakładce Sieć ograniczenie „Slow 4G” (lub „Wolne 3G”) i zrób zrzut stanu „Ładowanie...”. Następnie wybierz „Offline” i zrób zrzut komunikatu błędu.
3. Zrób zrzut działającej aplikacji z wykresem EUR i przelicznikiem. Spakuj folder.

W sprawozdaniu umieść

- ☐ opis formatu odpowiedzi API (tablica/obiekt, nazwy pól) i odpowiedź dla waluty XYZ
- ☐ dane żądania z zakładki Sieć (kod, Content-Type, rozmiar, czas, nagłówki CORS)
- ☐ zrzuty: stan ładowania, komunikat błędu, aplikacja z wykresem
- ☐ kod funkcji `pobierzJSON` i `wczytajTabele`
- ☐ spakowany folder `nazwisko_imie_z04`

Pytania kontrolne

1. Dlaczego `fetch` nie zgłasza wyjątku przy odpowiedzi 404 i co trzeba sprawdzić samemu?
2. Do czego służy blok `finally` w funkcji `wczytajTabele`?
3. Czym jest CORS i który nagłówek odpowiedzi API NBP pozwala stronie z innej domeny odczytać dane?
4. Co by się stało, gdyby nie wywołać `destroy()` przed utworzeniem nowego wykresu na tym samym elemencie canvas?

Kryteria oceny

Kryterium	Punkty
Analiza API: format odpowiedzi, kod dla XYZ, dane z zakładki Sieć	3
<code>pobierzJSON</code> z kontrolą <code>response.ok</code> , <code>async/await</code> , <code>try/catch/finally</code> , stan „Ładowanie...”	5
Tabela kursów z formatowaniem i filtrowaniem, data i numer tabeli, tryb offline	4
Wykres <code>Chart.js</code> dla wybranej waluty z obsługą błędów i <code>destroy()</code>	4
Przelicznik walut	1
Zrzuty (ładowanie, błąd, wykres) i odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

PHP — pierwsze skrypty

Czas: 2 × 45 min

praca indywidualna

Cel: Uruchomienie pierwszych skryptów PHP na serwerze Apache z XAMPP: zmienne i typy, echo, instrukcje warunkowe i match, pętle generujące tabelę i kalendarz oraz czytanie komunikatów o błędach.

Po zajęciach potrafisz:

- uruchamia skrypty PHP z folderu htdocs przez adres `http://localhost`
- używa zmiennych różnych typów, operatorów, `echo` i skróconego `<?= ?>`
- stosuje `if/elseif/else` i wyrażenie `match`
- generuje tabelę HTML pętlami `for` i `while`
- odczytuje komunikat o błędzie PHP (rodzaj, plik, wiersz) i poprawia kod

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- folder `r2_05_start` z Pliki_do_zadan (`styl.css`, `bledy.php`)

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
apachefriends.org/download.html
- **Visual Studio Code** — opcjonalnie rozszerzenie PHP Intelephense
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przeglądnij wykład 2.04. Sprawdź, czy port 80 nie jest zajęty (XAMPP Control Panel → Netstat).

Bezpieczeństwo

- Plik z `phpinfo()` i włączone `display_errors` są tylko do pracy na własnym komputerze — na serwerze publicznym ujawniają konfigurację. Usuń `info.php` po wykonaniu polecenia.
- Nie wyłączaj zapory Windows, aby Apache był dostępny z sieci — pracuj na `localhost`.

Przebieg zadania

1 Serwer i pierwszy skrypt

Pliki PHP nie działają po otwarciu dwuklikiem — wykonuje je serwer Apache z modułem PHP, a przeglądarka dostaje gotowy HTML.

1. Uruchom Apache w XAMPP Control Panel. Utwórz folder `C:\xampp\htdocs\nazwisko_z05` i skopiuj do niego pliki z `r2_05_start`.
2. Utwórz `info.php` z wywołaniem `phpinfo()`; i otwórz `http://localhost/nazwisko_z05/info.php`. Zapisz: wersję PHP, ścieżkę pliku `php.ini` (Loaded Configuration File), wartości `display_errors` i `date.timezone`. Następnie usuń plik.
3. Otwórz plik `.php` dwuklikiem w przeglądarce (adres `file:///...`) i porównaj z adresem przez `localhost`. Sprawdź Ctrl+U, że do przeglądarki nie trafia kod PHP.

2 Zmienne, typy i powitanie (index.php)

Na początku skryptu ustaw strefę czasową: `date_default_timezone_set('Europe/Warsaw');`. Dołącz `styl.css` i dodaj menu z odnośnikami do trzech podstron.

1. Utwórz zmienne `$imie` (tekst), `$klasa` (liczba całkowita), `$srednia` (liczba z przecinkiem), `$zdany` (logiczna). Wyświetl zdanie z ich wartościami — raz z operatorem `.`, raz w cudzysłowie z interpolacją — oraz typ każdej zmiennej funkcją `gettype()`. Co wyświetla `echo true`?
2. Wyrażeniem `match (true)` dobierz powitanie do godziny (`int`) `date('G')`: do 5 — „Dobrej nocy”, do 12 — „Dzień dobry”, do 18 — „Miłego popołudnia”, później — „Dobry wieczór”. Wyświetl je z imieniem, godziną i datą w formacie `21.10.2026`.
3. Instrukcją `if/elseif/else` opisz średnią (od 4,75 — „świadectwo z wyróżnieniem”, od 3,0 — „dobra średnia”, niżej — „trzeba popracować”). Średnią wyświetl przez `number_format` z przecinkiem.
4. Wyświetl wyniki: `17 / 5`, `intdiv(17, 5)`, `17 % 5`, `2 ** 10` i stałą `PHP_VERSION`.

3 Pętle: tabliczka i kalendarz

Do mieszania PHP z HTML możesz użyć składni alternatywnej `for (...): ?> ... <?php endfor;`.

1. W `tabliczka.php` wygeneruj tabliczkę mnożenia `$n × $n` (`$n = 10`) z nagłówkami wierszy i kolumn (`th`). Komórki na przekątnej (kwadraty) oznacz klasą `kwadrat`. Sprawdź, że zmiana `$n` na 12 powiększa tabelę.
2. Pod tabelą oblicz pętlą `while` sumę kwadratów od 1 do `$n` i wyświetl ją.
3. W `kalendarz.php` wygeneruj kalendarz bieżącego miesiąca od poniedziałku: nazwa miesiąca po polsku z tablicy, liczba dni z `date('t', ...)`, dzień tygodnia pierwszego dnia z `date('N', mktime(0, 0, 0, $miesiac, 1, $rok))`, puste komórki przed pierwszym dniem i po ostatnim. Soboty i niedziele — klasa `weekend`, dzisiejszy dzień — klasa `dzis`.

4 Czytanie błędów (bledy.php)

Plik ma cztery błędy. Uruchamiaj go po każdej poprawce — PHP zgłasza błędy po kolei.

1. Dla każdego komunikatu zapisz w tabeli: rodzaj (Parse error, Warning, Fatal error), treść, numer wiersza i poprawkę. Nie usuwaj kodu — popraw go (np. dzielenie wykonaj tylko, gdy liczba osób jest większa od 0; brakującą funkcję `oblicz_rabat` napisz: 5% rabatu od kwoty co najmniej 50 zł).
2. Zrób zrzuty ekranu: jednego komunikatu o błędzie, `index.php`, `tabliczka.php` i `kalendarz.php`. Spakuj folder.

W sprawozdaniu umieść

- ☐ dane z `phpinfo()`: wersja PHP, plik `php.ini`, `display_errors`, `date.timezone`
- ☐ zrzuty ekranu trzech podstron i jednego komunikatu o błędzie

- ☐ tabela błędów z `bledy.php` (rodzaj, treść, wiersz, poprawka)
- ☐ kod generujący kalendarz
- ☐ spakowany folder `nazwisko_z05`

Pytania kontrolne

1. Dlaczego plik PHP otwarty przez `file:///` nie działa, a przez `http://localhost/` działa? Co widać w źródle strony (Ctrl+U)?
2. Czym różni się Warning od Fatal error — czy skrypt wykonuje się dalej?
3. Czym różni się `match` od `switch` (porównanie, zwracanie wartości, brak `break`)?
4. Dlaczego na serwerze produkcyjnym `display_errors` powinno być wyłączone?

Kryteria oceny

Kryterium	Punkty
Serwer, phpinfo (4 informacje), porównanie <code>file://</code> i <code>localhost</code>	3
<code>index.php</code> : zmienne 4 typów, <code>gettype</code> , interpolacja, <code>match</code> dla 4 pór dnia, <code>if/elseif</code> , operatory	5
Tabliczka mnożenia z nagłówkami, wyróżnioną przekątną i sumą (<code>while</code>)	4
Kalendarz: poprawny pierwszy dzień, puste komórki, weekendy, dzień dzisiejszy	4
Tabela 4 błędów z poprawkami, zrzuty, odpowiedzi na pytania	4
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

PHP — tablice, funkcje i szablon strony

Czas: 2 × 45 min

praca indywidualna

Cel: Wyświetlanie danych z tablic asocjacyjnych w tabeli HTML, pisanie własnych funkcji z typami parametrów, użycie funkcji tablicowych oraz zbudowanie kilku podstron ze wspólnych plików dołączanych z wyróżnionym aktywnym odnośnikiem.

Po zajęciach potrafisz:

- przegląda tablice asocjacyjne pętlą `foreach` i wyświetla je w tabeli
- pisze funkcje z typami parametrów, wartością domyślną i typem zwracanym
- stosuje funkcje tablicowe: `count`, `array_column`, `array_filter`, `array_map`, `in_array`, `implode` / `explode`, `usort`
- dzieli stronę na pliki `include` / `require` i wyróżnia aktywny link w menu
- korzysta z dat (`date`, `DateTime`)

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- folder `r2_06_start` z Pliki_do_zadan (`dane.php`, `styl.css`)

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
[apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code** — opcjonalnie rozszerzenie PHP Intelephense
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przeglądnij wykład 2.05 (tablice, funkcje, include/require).

Bezpieczeństwo

- Wartość z adresu (`$_GET['sort']`) przyjmuj tylko z listy dozwolonych (`in_array(..., true)`), nigdy nie wstawiaj jej bez sprawdzenia.
- Pliki dołączane (`inc/`) nie powinny być otwierane bezpośrednio — sprawdź, co pokazuje `http://localhost/.../inc/stopka.php`, i opisz to w sprawozdaniu.

Przebieg zadania

1 Funkcje (funkcje.php)

Skopiuj pliki z `r2_06_start` do `C:\xampp\htdocs\nazwisko_z06`. Na początku `funkcje.php` dodaj `declare(strict_types=1);`.

1. Napisz `cenaBrutto(float $netto, int $vat = 23): float` — cena brutto zaokrąglona do 2 miejsc (`round`).
2. Napisz `zl(float $kwota): string` — format „1 234,50 zł” (`number_format`).
3. Napisz `kategorie(array $produkty): array` — unikalne kategorie posortowane alfabetycznie (`array_column`, `array_unique`, `sort`).
4. Napisz `aktywny(string $plik, string $biezacy): string` — zwraca `class="aktywny"`, gdy nazwy plików są równe, w przeciwnym razie pusty tekst.

2 Szablon: nagłówek, menu, stopka

Każda podstrona zaczyna się od `$tytul = '...'; require 'inc/naglowek.php';` i kończy `require 'inc/stopka.php';`.

```
inc/naglowek.php - doctype, head z <title>, header, include menu.php, <main><h2>
inc/menu.php     - tablica ['index.php' => 'Start', ...] i pętla foreach
inc/stopka.php   - </main>, footer z rokiem date('Y'), </body></html>
```

1. W `inc/naglowek.php` dołącz `funkcje.php` i `dane.php` przez `require_once __DIR__ . '/../...'`, ustaw strefę czasową i nazwę bieżącego pliku: `$biezacy = basename($_SERVER['SCRIPT_NAME']);`. Tytuł karty: „<tytuł> — Kawiarnia Ziarno”.
2. W `inc/menu.php` zbuduj menu z tablicy asocjacyjnej (plik → opis) pętlą `foreach`; aktywny link wyróżnij funkcją `aktywny()`.
3. W `inc/stopka.php` wyświetl bieżący rok, liczbę pozycji w karcie (`count`) i swoje imię i nazwisko.

3 Podstrony

1. `index.php`: dzisiaj data (`DateTime`, format `d.m.Y`) i liczba dni do najbliższej Wigilii (`diff()->days`); liczba dostępnych pozycji (`array_filter`); najniższa, najwyższa i średnia cena brutto dostępnych pozycji (`array_map`, `min`, `max`, `array_sum`); lista kategorii (`implode`); odpowiedź, czy w karcie jest „Szarlotka” (`in_array`).
2. `karta.php`: tabela wszystkich pozycji — Lp., nazwa, kategoria z wielkiej litery, netto, VAT, brutto (pogrubione), tagi rozdzielone `explode` i połączone znakiem „ · ”. Pozycje niedostępne — klasa `niedostepny`.
3. W `karta.php` dodaj sortowanie odnośnikami `?sort=nazwa` i `?sort=cena` (`usort` z operatorem `<=>` na kopii tablicy). Nieznaną wartość `sort` zamień na „domyslnie”.
4. `kontakt.php`: adres i tabela godzin otwarcia z tablicy `$godziny`; dzisiejszy dzień (numer z `date('N')`) wyróżnij klasą `dzis`, a dzień bez godzin opisz jako „nieczynne”. Nazwy dni z wielkiej litery — sprawdź, co robi `ucfirst('środa')`, i użyj `mb_convert_case(..., MB_CASE_TITLE)`.

4 Test szablonu

1. Dodaj do menu czwartą pozycję (np. `onas.php` — „O nas”) i utwórz tę podstronę. Ile plików musiałeś zmienić, aby link pojawił się na wszystkich stronach?
2. Zrób zrzuty ekranu trzech podstron (karta posortowana według ceny) i spakuj folder.

W sprawozdaniu umieść

- ☐ kod funkcji z `funkcje.php` i pliku `inc/menu.php`
- ☐ zrzuty ekranu podstron z wyróżnionym aktywnym linkiem, karta posortowana według ceny
- ☐ wyniki z `index.php` (statystyki cen) i opis działania `ucfirst` dla „środa”

- ☐ odpowiedź: ile plików zmieniono przy dodawaniu podstrony „O nas”
- ☐ spakowany folder `nazwisko_z06`

Pytania kontrolne

1. Czym różni się `include` od `require` i po co jest wersja `_once`?
2. Dlaczego w `inc/naglowek.php` używa się `__DIR__` w ścieżkach?
3. Co zwraca operator `<=>` i jak `usort` korzysta z tego wyniku?
4. Dlaczego `ucfirst('środa')` nie zmienia pierwszej litery?

Kryteria oceny

Kryterium	Punkty
Cztery funkcje z typami, wartością domyślną i poprawnym formatowaniem kwot	4
Szablon: nagłówek, menu z tablicy, stopka; aktywny link na każdej podstronie	5
index.php: data, dni do Wigilii, statystyki cen, kategorie, in_array	3
karta.php: tabela z brutto, tagami, niedostępnymi; sortowanie z listą dozwolonych wartości	4
kontakt.php: godziny, dzisiejszy dzień, „nieczynne”, poprawne wielkie litery	2
Zrzuty, test czwartej podstrony, odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

PHP — obsługa formularzy

Czas: 2 × 45 min

praca indywidualna

Cel: Odbieranie danych z formularzy metodami GET i POST, walidacja po stronie serwera z komunikatami przy polach, ponowne wypełnianie pól z bezpiecznym wypisywaniem (`htmlspecialchars`) oraz zastosowanie wzorca Post/Redirect/Get.

Po zajęciach potrafisz:

- dobiera metodę GET lub POST do rodzaju formularza
- sprawdza dane funkcjami `isset`, `filter_input`, `filter_var` z zakresami
- wyświetla komunikaty błędów przy polach i zachowuje wpisane wartości
- wypisuje dane użytkownika przez `htmlspecialchars`
- stosuje wzorzec PRG (`header('Location: ...')` i `exit`)

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- folder `r2_07_start` z Pliki_do_zadan (`styl.css`, `podroz.html`, `rata.html`)

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
apachefriends.org/download.html
- **Visual Studio Code**
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przeglądaj wykład 2.06 (`$_GET`, `$_POST`, `filter_var`, PRG).

Bezpieczeństwo

- Każdą wartość od użytkownika wypisuj przez funkcję `e()` (`htmlspecialchars` z `ENT_QUOTES`) — także w atrybucie `value`.
- Plik `zapytania.csv` zawiera dane osobowe (imię, e-mail) — używaj tylko danych testowych i usuń plik po zajęciach. Na prawdziwym serwerze taki plik nie może leżeć w katalogu dostępnym z przeglądarki.

Przebieg zadania

1 Funkcje pomocnicze

Skopiuj pliki do `C:\xampp\htdocs\nazwisko_z07`. Zmień nazwy `podroz.html` → `podroz.php`, `rata.html` → `rata.php`. Utwórz `funkcje.php`.

```
function e(?string $tekst): string {
    return htmlspecialchars($tekst ?? '', ENT_QUOTES, 'UTF-8');
}
// liczba("6,5", 2, 30) → 6.5;  liczba("abc", 2, 30) → null
function liczba(?string $wartosc, float $min, float $max): ?float { ... }
```

1. Przepisz funkcję `e()` i napisz `liczba()`: zamień przecinek na kropkę, sprawdź `filter_var(..., FILTER_VALIDATE_FLOAT, ['options' => ['min_range' => $min, 'max_range' => $max]])`, zwróć `null`, gdy wynik to `false`.
2. Napisz `zl(float $kwota): string` oraz `bładPola(array $bledy, string $pole): string`, która zwraca `<p class="bład">...</p>` albo pusty tekst.

2 Koszt podróży — GET

Formularz GET nadaje się do obliczeń, które nic nie zmieniają na serwerze: wynik można zapisać jako zakładkę lub wystać linkiem.

1. Formularz uznaj za wysłany, gdy `isset($_GET['dystans'])`. Sprawdź: dystans 1–5000 km, spalanie 2–30 l/100 km, cena 3–15 zł/l (wszystkie z przecinkiem lub kropką), liczba osób całkowita 1–9 — przez `filter_input(INPUT_GET, 'osoby', FILTER_VALIDATE_INT, ...)`.
2. Błędy zbieraj w tablicy `$bledy['pole'] = 'komunikat'`; wyświetl je pod polami, a błędne pola oznacz klasą `zle`. Po wysłaniu pola zachowują wpisane wartości, pole „W obie strony” — zaznaczenie.
3. Gdy dane są poprawne, oblicz: liczbę km (×2 dla podróży w obie strony), litry, koszt i koszt na osobę. Test: 320 km, 6,5 l, 6,29 zł, 3 osoby, w obie strony. Skopiuj adres z paska i otwórz w nowej karcie — co się stało?

3 Rata kredytu — POST

Rata równa: $R = K \cdot r / (1 - (1 + r)^{-n})$, gdzie K — kwota, r — oprocentowanie roczne / 100 / 12, n — liczba rat. Dla $r = 0$: $R = K / n$.

1. Obsłuż formularz, gdy `$_SERVER['REQUEST_METHOD'] === 'POST'`. Sprawdź: kwota 1000–500 000 zł, oprocentowanie 0–30%, liczba rat całkowita 6–360. Pokaż ratę, łączną kwotę do spłaty i odsetki.
2. Wpisz w pole kwoty tekst z cudzysłowem, np. `30 000 "zł"`. Obejrzyj źródło strony (Ctrl+U) i zapisz, jak wygląda atrybut `value`.
3. Test: 30 000 zł, 8,5%, 36 rat; drugi test: oprocentowanie 0%.

4 Zapytanie do doradcy i PRG

Formularz ma dwa przyciski `submit` z `name="akcja"` i wartościami `oblicz` oraz `zapytanie`.

```
$plik = fopen(__DIR__ . '/zapytania.csv', 'a');
fputcsv($plik, $wiersz, ';');
fclose($plik);
header('Location: rata.php?wyslano=1', true, 303);
exit;
```

1. Dla akcji `zapytanie` sprawdź dodatkowo imię (min. 3 znaki, `mb_strlen`) i e-mail (`FILTER_VALIDATE_EMAIL`). Jeśli wszystko jest poprawne, dopisz wiersz do `zapytania.csv` (data, imię, e-mail, kwota, procent, raty, rata) i przekieruj przeglądarkę (kod powyżej).
2. Gdy w adresie jest `wyslano=1`, pokaż komunikat „Dziękujemy! Zapytanie zostało zapisane”. Odśwież stronę (F5) — czy wiersz w CSV się powtórzył? Usuń na chwilę `header` i `exit`, wyślij zapytanie, naciśnij F5 i opisz różnicę. Przywróć PRG.
3. W F12 → Sieć obejrzyj żądanie POST: kod odpowiedzi 303, nagłówek `Location` i dane formularza (Ładunek/Payload). Zrób zrzuty i spakuj folder (bez pliku CSV).

W sprawozdaniu umieść

- ☐ tabela testów obu kalkulatorów (dane wejściowe, oczekiwany i otrzymany wynik)
- ☐ adres URL z wynikiem kalkulatora podróży i wyjaśnienie, dlaczego działa w nowej karcie
- ☐ fragment źródła strony z atrybutem `value` dla tekstu z cudzysłowem
- ☐ opis zachowania F5 z PRG i bez PRG oraz zrzut zakładki Sieć (303, Location)
- ☐ spakowany folder `nazwisko_z07`

Pytania kontrolne

1. Kiedy wybrać GET, a kiedy POST? Podaj po jednym przykładzie z tego zadania.
2. Dlaczego walidacja na serwerze jest konieczna, nawet gdy formularz ma atrybuty HTML i walidację w JS?
3. Co robi `htmlspecialchars` i dlaczego potrzebna jest flaga `ENT_QUOTES`?
4. Jaki problem rozwiązuje wzorzec PRG i dlaczego po `header('Location: ...')` piszemy `exit`?

Kryteria oceny

Kryterium	Punkty
Funkcje <code>e()</code> , <code>liczba()</code> , <code>zł()</code> , <code>bladPola()</code>	3
Kalkulator GET: 4 reguły walidacji, komunikaty, zachowane wartości, poprawny wynik	5
Kalkulator POST: walidacja, wzór raty (także 0%), odsetki, <code>htmlspecialchars</code> w <code>value</code>	5
Zapytanie: walidacja imienia i e-maila, zapis CSV, PRG z komunikatem	4
Testy, analiza F5 i zakładki Sieć, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

PHP i MySQL — MySQLi jak na egzaminie

Czas: 2 × 45 min

praca indywidualna

Cel: Połączenie strony PHP z bazą MariaDB/MySQL rozszerzeniem MySQLi w stylu egzaminu INF.03: lista i tabela z wyników zapytań, zapytanie z parametrem z formularza przez `mysqli_prepare` i dodanie rekordu z formularza.

Po zajęciach potrafisz:

- importuje bazę z pliku i pisze kwerendy SQL do zadania egzaminacyjnego
- łączy się z bazą `mysqli_connect`, ustawia kodowanie i zamyka połączenie
- wyświetla wyniki w pętli `mysqli_fetch_assoc` / `row` / `array` i liczy wiersze `mysqli_num_rows`
- wykonuje zapytania z parametrami przez `mysqli_prepare` i `mysqli_stmt_bind_param`
- dodaje rekord z formularza i odczytuje jego identyfikator `mysqli_insert_id`

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- plik `r2_08_schronisko.sql` i folder `r2_08_start` z Pliki_do_zadan

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
apachefriends.org/download.html
- **Visual Studio Code**
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przeglądnij wykład 2.07 (`mysqli_connect`, `mysqli_query`, `mysqli_fetch_*`, `mysqli_prepare`).

Bezpieczeństwo

- Wartości z formularza przekazuj do SQL wyłącznie jako parametry zapytania przygotowanego — nigdy nie doklejaj ich do tekstu zapytania.
- Konto `root` bez hasła jest dopuszczalne tylko na lokalnym XAMPP; nie udostępniaj serwera MySQL w sieci.
- Dane wpisywane w formularzu adopcyjnym muszą być fikcyjne.

Przebieg zadania

1 Baza danych i kwerendy

Uruchom Apache i MySQL w XAMPP. Zaimportuj `r2_08_schronisko.sql` (phpMyAdmin → Import). Skrypt tworzy bazę `schronisko` od nowa.

- Obejrzyj strukturę tabel `zwierzeta` i `zgloszenia` oraz relację między nimi. Zapisz, które pola mogą mieć wartość NULL.
- Zapisz w pliku `kwerendy.txt` i wykonaj: Z1 — imię, rasa i wiek psów czekających na adopcję (`adoptowany = 0`), od najstarszego; Z2 — liczba zwierząt każdego gatunku; Z3 — osoba, data zgłoszenia i imię zwierzęcia dla wszystkich zgłoszeń (JOIN); Z4 — oznaczenie zwierzęcia o imieniu Burek jako adoptowanego (UPDATE). Zrób zrzut wyniku każdej kwerendy.
- Po Z4 zaimportuj bazę ponownie, aby witryna pracowała na danych wyjściowych.

2 Witryna i skrypt 1

Skopiuj pliki z `r2_08_start` do `C:\xampp\htdocs\schronisko` i zmień nazwę `schronisko.html` na `schronisko.php`. W stopce zamiast numeru PESEL (jak na egzaminie) wpisz klasę i swój numer z dziennika, np. `2T-14`.

```
$polaczenie = mysqli_connect('localhost', 'root', '', 'schronisko');
mysqli_set_charset($polaczenie, 'utf8mb4');
// ... skrypty ...
mysqli_close($polaczenie);
```

- Na początku pliku połącz się z bazą; przy błędzie zakończ skrypt komunikatem „Brak połączenia z bazą danych.” (`die`). Na końcu strony zamknij połączenie.
- Skrypt 1 (panel lewy): wyślij zapytanie o imię, gatunek i wiek zwierząt nieadoptowanych, posortowanych według daty przyjęcia. Wyświetl liczbę zwierząt (`mysqli_num_rows`), a pod nią listę numerowaną w formacie „Imię – gatunek (wiek lat)” (`mysqli_fetch_assoc`).

3 Skrypt 2 — wyszukiwanie z parametrem

- W panelu prawym dodaj formularz GET z listą `gatunek` (pies, kot, królik) i przyciskiem „Szukaj”. Po wysłaniu lista pokazuje wybrany gatunek (`selected`).
- Skrypt 2: jeśli formularz został wysłany, wykonaj przez `mysqli_prepare` zapytanie o imię, rasę, wiek, płeć i datę przyjęcia nieadoptowanych zwierząt wybranego gatunku (sortowanie po imieniu). Wynik pokaż w tabeli z nagłówkami (`mysqli_stmt_get_result`, `mysqli_fetch_row`).
- Zamiast NULL w kolumnie rasa wypisz „nieznana”, a płeć K/M zamień na „samica”/„samiec”. Gdy wynik jest pusty, wypisz „Brak zwierząt tego gatunku”.

4 Skrypt 3 — dodawanie rekordu

- W ramce „Zgłoszenie adopcyjne” dodaj formularz POST: lista zwierząt nieadoptowanych pobrana z bazy (`value` = id, tekst „Imię (gatunek)”, `mysqli_fetch_array`), pole imienia i nazwiska, pole telefonu (9 cyfr) i przycisk „Wyślij zgłoszenie”.
- Skrypt 3: po wysłaniu sprawdź dane (id liczbą całkowitą, osoba min. 3 znaki, telefon 9 cyfr po usunięciu spacji) i dodaj rekord do `zgloszenia` z bieżącą datą (`CURDATE()`) zapytaniem przygotowanym (typy `iss`). Wyświetl „Dziękujemy! Zgłoszenie nr N zostało zapisane.” z numerem z `mysqli_insert_id`.
- Dodaj zgłoszenie testowe i sprawdź w phpMyAdmin, że rekord jest w tabeli. Zrób zrzut całej strony po wyszukaniu gatunku „królik” i spakuj folder z plikiem `kwerendy.txt`.

W sprawozdaniu umieść

- ☐ plik `kwerendy.txt` i zrzuty wyników Z1–Z4

- ☐ zrzut strony po wyszukaniu gatunku „królik” oraz po dodaniu zgłoszenia
- ☐ zrzut tabeli `zgłoszenia` w phpMyAdmin z nowym rekordem
- ☐ kod skryptów 1–3
- ☐ spakowany folder `schronisko`

Pytania kontrolne

1. Czym różnią się `mysqli_fetch_assoc`, `mysqli_fetch_row` i `mysqli_fetch_array`?
2. Dlaczego wartość z formularza przekazujemy przez `mysqli_prepare` i `bind_param`, a nie doklejamy do zapytania?
3. Co oznaczają litery `i` i `s` w `mysqli_stmt_bind_param`?
4. Po co wywołujemy `mysqli_set_charset($polaczenie, 'utf8mb4')`? Co widać bez tego przy polskich znakach?

Kryteria oceny

Kryterium	Punkty
Kwerendy Z1–Z4 poprawne, zrzuty wyników	4
Połączenie z bazą, kodowanie, obsługa błędu połączenia, zamknięcie połączenia	2
Skrypt 1: liczba zwierząt i lista w wymaganym formacie i kolejności	3
Skrypt 2: formularz GET z zapamiętanym wyborem, <code>mysqli_prepare</code> , tabela, NULL i płeć, brak wyników	5
Skrypt 3: lista z bazy, walidacja, INSERT przygotowany, komunikat z numerem zgłoszenia	4
Zrzuty i odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

PHP i PDO — CRUD: katalog filmów

Czas: 2 × 45 min

praca indywidualna lub w parach

Cel: Zbudowanie aplikacji z pełnym zestawem operacji CRUD (lista, dodawanie, edycja, usuwanie) na bazie MariaDB/MySQL z użyciem PDO, zapytań przygotowanych, obsługi wyjątków, potwierdzenia usunięcia i komunikatów flash.

Po zajęciach potrafisz:

- tworzy połączenie PDO z DSN i opcjami (ERRMODE_EXCEPTION, FETCH_ASSOC)
- wykonuje zapytania przygotowane z parametrami nazwanymi i pozycyjnymi
- realizuje operacje SELECT, INSERT, UPDATE, DELETE z formularzy i odczytuje `lastInsertId`
- stosuje potwierdzenie usunięcia metodą POST i wzorzec PRG z komunikatem flash
- oddziela konfigurację połączenia od kodu stron

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- plik `r2_09_filmoteka.sql` i folder `r2_09_start` z Pliki_do_zadan

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
apachefriends.org/download.html
- **Visual Studio Code**
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przejrzyj wykład 2.08 (PDO, prepare/execute, wyjątki).
- Zaimportuj `r2_09_filmoteka.sql` w phpMyAdmin.

Bezpieczeństwo

- Wszystkie wartości od użytkownika przekazuj jako parametry `execute()`. Nazw kolumn do ORDER BY nie da się przekazać parametrem — wybieraj je z tablicy dozwolonych wartości.
- Usuwanie wykonuj tylko po potwierdzeniu wysłanym metodą POST — link GET nie może niczego usuwać.
- Plik `config.php` z hasłem nie trafia do publicznego repozytorium; na serwerze umieszcza się go poza katalogiem publicznym.

Przebieg zadania

1 Struktura i połączenie

Utwórz projekt `C:\xampp\htdocs\filmoteka` ze stylem z `r2_09_start` i folderem `inc`.

```
filmoteka/  
  index.php  dodaj.php  edytuj.php  usun.php  styl.css  
  inc/config.php    - return ['dsn' =>  
'mysql:host=localhost;dbname=filmoteka;charset=utf8mb4', ...]  
  inc/db.php        - session_start(), db(), e(), flash(), pokazFlash(), przekieruj()  
  inc/gora.php  inc/dol.php  inc/formularz.php
```

- W `inc/config.php` zwróć (`return`) tablicę z DSN, użytkownikiem i hasłem. W `inc/db.php` napisz funkcję `db(): PDO`, która tworzy połączenie raz (zmienna `static`) z opcjami `ERRMODE_EXCEPTION`, `FETCH_ASSOC` i `ATTR_EMULATE_PREPARES => false`.
- Dopisz funkcje: `e()` (htmlspecialchars), `flash($tekst, $typ)` zapisującą komunikat w `$_SESSION['flash']`, `pokazFlash()` wyświetlającą go raz i usuwającą z sesji, `przekieruj($adres)` z `header('Location: ...', true, 303)` i `exit`.
- Zmień w konfiguracji nazwę bazy na błędną, otwórz stronę i zapisz treść wyjątku PDOException. Przywróć poprawną nazwę.

2 Lista (Read)

- `index.php`: tabela filmów (tytuł, reżyser, rok, gatunek, ocena lub „—”, znacznik obejrzany) z linkami „Edytuj” i „Usuń” (`?id=`). Nad tabelą statystyka jednym zapytaniem: liczba filmów, liczba obejrzanych, średnia ocena (`ROUND(AVG(ocena), 2)`).
- Dodaj sortowanie `?sort=tytuł|rok|ocena` — klauzulę `ORDER BY` wybieraj z tablicy dozwolonych wartości, domyślnie po tytule.

3 Dodawanie i edycja (Create, Update)

Oba formularze używają wspólnego pliku `inc/formularz.php`. Ocena jest opcjonalna — puste pole zapisz jako NULL.

- Napisz walidację `sprawdzFilm(array $f): array`: tytuł 1–120 znaków, reżyser min. 3 znaki, rok całkowity od 1895 do bieżącego + 2, gatunek niepusty, ocena pusta albo 1–10 (z przecinkiem lub kropką). Błędy pokazuj przy polach, a wpisane wartości zachowuj.
- `dodaj.php`: po poprawnej walidacji `INSERT` z parametrami nazwanymi (`:tytuł`, ...), komunikat `flash „Dodano film „...” (id N).”` z `lastInsertId()` i przekierowanie na listę.
- `edytuj.php?id=N`: pobierz film (`prepare` + `execute([$id])` + `fetch`); gdy nie istnieje — `flash` z błędem i powrót do listy. Po wysłaniu `UPDATE` i `flash „Zapisano zmiany...”`.

4 Usuwanie (Delete) i testy

- `usun.php?id=N` (GET) pokazuje pytanie „Czy na pewno usunąć film ... (rok)?” z przyciskiem „Tak, usuń” (formularz POST) i linkiem „Nie, wróć do listy”. Dopiero POST wykonuje `DELETE`, ustawia `flash` i przekierowuje.
- Wykonaj testy z tabeli w sprawozdaniu: pusty formularz, dodanie filmu (np. „Kiler”, 1997), F5 po dodaniu, edycja oceny „Dnia świra”, usunięcie „Psów”, edycja nieistniejącego id 999. Zrób zrzuty listy z komunikatem `flash` i ekranu potwierdzenia.

W sprawozdaniu umieść

- ☐ struktura plików projektu i kod `inc/db.php`
- ☐ treść wyjątku PDOException przy błędnej nazwie bazy
- ☐ tabela testów CRUD z wynikami

- ❑ zrzuty: lista z komunikatem flash, formularz z błędami, ekran potwierdzenia usunięcia
- ❑ spakowany folder `filmoteka` (lub link do repozytorium bez hasła w config.php)

Pytania kontrolne

1. Co daje opcja `PDO::ERRMODE_EXCEPTION` w porównaniu z domyślnym zachowaniem przy błędzie zapytania?
2. Dlaczego usuwanie nie może odbywać się po kliknięciu zwykłego linku (GET)?
3. Dlaczego nazwy kolumny do sortowania nie można przekazać jako parametru `:kolumna`?
4. Czy jest komunikat flash i dlaczego musi przetrwać przekierowanie?

Kryteria oceny

Kryterium	Punkty
Połączenie PDO w osobnych plikach (config, db), opcje, funkcje pomocnicze	3
Lista ze statystyką i bezpiecznym sortowaniem	3
Dodawanie: walidacja 5 pól, zachowane wartości, INSERT przygotowany, NULL dla pustej oceny, lastInsertId	5
Edycja: pobranie rekordu, obsługa brakującego id, UPDATE przygotowany	3
Usuwanie z potwierdzeniem POST, flash i PRG we wszystkich operacjach	3
Testy, zrzuty, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Zadanie kontrolne: PHP + baza

Czas: 2 × 45 min

praca samodzielna (sprawdzian praktyczny), bez internetu; dozwolone: dokumentacja PHP offline i notatki własne — według decyzji nauczyciela

Cel: Samodzielne wykonanie w 90 minut zadania w stylu egzaminu INF.03: kwerendy SQL, witryna z układem blokowym i stylami według specyfikacji oraz skrypty PHP wyświetlające, wyszukujące i dodające dane w bazie.

Po zajęciach potrafisz:

- pisze kwerendy SELECT z warunkiem, grupowaniem i UPDATE według treści polecenia
- tworzy witrynę z układem blokowym i stylem zgodnym ze specyfikacją
- wyświetla dane z bazy w tabeli HTML
- obsługuje formularz wyszukiwania z zapytaniem przygotowanym i operatorem LIKE
- dodaje rekord do bazy z formularza

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- plik `r2_10_biblioteka.sql` i folder `r2_10_start` (`ksiazki.png`) z Pliki_do_zadan

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
apachefriends.org/download.html
- **Visual Studio Code lub Notepad++** —
Notepad++: <https://notepad-plus-plus.org/code.visualstudio.com>
- **Przeglądarka Chrome, Edge lub Firefox** —
narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Powtórz zadania 5–9 i strukturę arkusza INF.03 (baza, witryna, skrypt).

Bezpieczeństwo

- Dane z formularzy przekazuj do bazy przez zapytania przygotowane (MySQLi lub PDO — do wyboru).
- Pracuj wyłącznie na bazie `biblioteka` na własnym komputerze.

Przebieg zadania

1 Baza danych (4 pkt)

Zaimportuj `r2_10_biblioteka.sql`. Utwórz folder `C:\xampp\htdocs\nazwisko_kontrolne` i w nim plik `kwerendy.txt`. Zrób zrzuty wyników kwerend (`kw1.png` ... `kw4.png`).

1. Z1 — tytuły i autorzy książek z gatunku fantastyka, alfabetycznie według tytułu.
2. Z2 — gatunek i łączna liczba egzemplarzy w każdym gatunku (kolumna `egzemplarzy`).
3. Z3 — tytuły książek, których wszystkie egzemplarze są wypożyczone.
4. Z4 — zwiększenie o 2 liczby egzemplarzy książki „Pan Tadeusz”.

2 Witryna i styl

Plik `biblioteka.php` (UTF-8, język polski, tytuł karty „Biblioteka szkolna”) i arkusz `styl.css`. Układ: baner, pod nim panel lewy i panel prawy obok siebie, na dole stopka.

1. Baner: tło `#3B5B7A`, biały tekst, wysokość 140 px; nagłówek `h1` „Biblioteka szkolna „Pod Lipą”” wyśrodkowany w pionie; obraz `ksiazki.png` wyrównany do prawej, wysokość 120 px, tekst alternatywny „Książki na półce”.
2. Panel lewy: szerokość 35%, wysokość 560 px, tło `#E8EEF4`, nagłówek `h2` „Dostępne teraz” i tabela (skrypt 1). Panel prawy: szerokość 65%, wysokość 560 px, nagłówki `h2` „Szukaj w katalogu” i „Dodaj książkę”.
3. Stopka: tło jak baner, biały tekst wyśrodkowany „Autor: ” i Twoja klasa z numerem z dziennika. Całość: krój Tahoma, tło strony `#F5F7FA`, nagłówki `h2` w kolorze `#3B5B7A`.
4. Tabela: obramowanie 1 px ciągłe w kolorze `#3B5B7A`, scalone krawędzie, komórki nagłówkowe z tłem `#3B5B7A` i białym tekstem. Przyciski formularzy: tło `#3B5B7A`, biały tekst, bez obramowania, po najechaniu kursorem tło `#22394F`.

3 Skrypty PHP

1. Skrypt 1 (panel lewy): tabela z kolumnami Tytuł, Autor, Wolne egz. dla książek, których nie wszystkie egzemplarze są wypożyczone (wolne = egzemplarze – wypożyczone), alfabetycznie według tytułu.
2. Skrypt 2 (panel prawy): formularz GET z polem `fraza` i przyciskiem „Szukaj”. Po wysłaniu niepustej frazy wyświetl „Znaleziono: N” i listę punktowaną „Tytuł — Autor (rok)” dla książek, których tytuł lub autor zawiera frazę, rosnąco według roku wydania. Wpisana fraza zostaje w polu.
3. Skrypt 3: formularz POST (tytuł, autor, rok wydania 1450–2030, gatunek z listy: powieść, fantastyka, dramat, poezja, wspomnienia, liczba egzemplarzy 1–50, domyślnie 1, przycisk „Dodaj”). Po wysłaniu poprawnych danych dodaj rekord i wyświetl „Dodano książkę „tytuł”.”, w przeciwnym razie „Uzupełnij poprawnie wszystkie pola.”.
4. Na końcu zamknij połączenie z bazą. Zrób zrzut strony po wyszukaniu frazy „lem” (`witryna.png`). Spakuj folder do ZIP.

W sprawozdaniu umieść

- ☐ plik `kwerendy.txt` i zrzuty `kw1.png` – `kw4.png`
- ☐ pliki `biblioteka.php`, `styl.css`, `ksiazki.png`
- ☐ zrzut `witryna.png` po wyszukaniu frazy „lem”
- ☐ krótka samoocena: które polecenia sprawiły trudność i ile czasu zajęły

Pytania kontrolne

1. Jak zbudować wzorzec LIKE dla frazy z formularza, aby był bezpieczny i działał z zapytaniem przygotowanym?

2. Dlaczego książki „Pan Tadeusz” nie ma w tabeli „Dostępne teraz” przed wykonaniem Z4, a po Z4 jest?
3. Która część zadania zajęła Ci najwięcej czasu i jak przyspieszysz ją na egzaminie?

Kryteria oceny

Kryterium	Punkty
Kwerendy Z1–Z4 poprawne (po 1 pkt) ze zrzutami	4
Struktura HTML: baner z obrazem i alt, panele, stopka, poprawne kodowanie i tytuł	3
CSS zgodny ze specyfikacją (kolory, wymiary, krój, tabela, przyciski z :hover)	4
Skrypt 1: zapytanie, kolumna wyliczana, kolejność, tabela	3
Skrypt 2: formularz GET, zapytanie przygotowane z LIKE, liczba wyników, lista, zachowana fraza	3
Skrypt 3: formularz, walidacja zakresów, INSERT przygotowany, komunikaty, zamknięcie połączenia	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Sesje i ciasteczka

Czas: 2 × 45 min

praca indywidualna

Cel: Zapamiętywanie stanu w bezstanowym protokole HTTP: licznik odwiedzin i preferencja motywu w ciasteczkach z właściwymi atrybutami oraz koszyk zakupów w sesji PHP, sprawdzane w narzędziach F12.

Po zajęciach potrafisz:

- ustawia, odczytuje i usuwa ciasteczka funkcją `setcookie` z atrybutami `expires`, `path`, `HttpOnly`, `SameSite`
- wyjaśnia znaczenie atrybutu `Secure` i powód jego pominięcia na `http://localhost`
- korzysta z sesji (`session_start`, `$_SESSION`) do przechowywania koszyka
- obsługuje dodawanie, zmniejszanie i czyszczenie koszyka metodą POST z wzorcem PRG
- odczytuje ciasteczka i nagłówki `Set-Cookie` w narzędziach F12

Sprzęt i materiały

- komputer z Windows 11 i XAMPP
- folder `r2_11_start` z Pliki_do_zadan (`produkty.php`, `styl.css`)

Oprogramowanie

- **XAMPP (Apache, PHP 8, MariaDB, phpMyAdmin)** — projekt w folderze `C:\xampp\htdocs`
[apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code**
code.visualstudio.com
- **Przeglądarka Chrome, Edge lub Firefox** — narzędzia deweloperskie F12 (Elementy, Konsola, Sieć, Aplikacja)

Przygotowanie przed zajęciami

- Przeglądnij wykład 2.09 (cookies, sesje, atrybuty ciasteczek, RODO).

Bezpieczeństwo

- Wartość ciasteczka pochodzi od przeglądarki i może być dowolna — przed użyciem sprawdzaj ją (lista dozwolonych wartości, `filter_var`).
- Nie wyświetlaj identyfikatora sesji na stronie i nie przekazuj go w adresie URL — kto go zna, może przejąć sesję.
- Ciasteczka preferencji (inne niż niezbędne) na prawdziwej stronie wymagają zgody użytkownika; w ćwiczeniu wystarczy informacja na stronie.

Przebieg zadania

1 Wspólny plik i sesja

Skopiuj pliki do `C:\xampp\htdocs\nazwisko_z11`. Utwórz `wspolne.php` dołączany na początku każdej strony — przed jakimkolwiek HTML (ciasteczka i sesja wysyłają nagłówki).

```
session_set_cookie_params(['lifetime' => 0, 'path' => '/', 'httponly' => true, 'samesite' => 'Lax']);
session_start();
require __DIR__ . '/produkty.php';
```

- W `wspolne.php` uruchom sesję jak wyżej, odczytaj motyw z ciasteczka `motyw` (dozwolone tylko `jasny` / `ciemny`, domyślnie `jasny`), koszyk z `$_SESSION['koszyk']` (tablica id → ilość) i liczbę sztuk (`array_sum`).
- Napisz funkcję `naglowek($tytul, $motyw, $sztuk)`, która wypisuje początek strony: `body` z klasą `ciemny` przy motywie ciemnym oraz menu Start, Sklep, „Koszyk (N)”.

2 Ciasteczka: licznik i motyw (index.php)

- Licznik odwiedzin: odczytaj `$_COOKIE['odwiedziny']` (liczba całkowita ≥ 0 , w przeciwnym razie 0), zwiększ o 1 i zapisz `setcookie('odwiedziny', ..., [opcje])` na 30 dni z `path`, `httponly` i `samesite = Lax`. Pierwsza wizyta: „Jesteś u nas pierwszy raz”, kolejne: „Odwiedzasz nas po raz N.”.
- Formularz POST z przyciskami „Jasny” i „Ciemny” zapisuje ciasteczko `motyw` na 365 dni, a następnie przekierowuje na `index.php` (PRG). Sprawdź, że motyw działa na obu podstronach i po zamknięciu przeglądarki.
- Przycisk „Usuń moje ciasteczka” usuwa `motyw` i `odwiedziny` (ta sama nazwa i ścieżka, data wygaśnięcia w przeszłości).
- Dodaj w opcjach `'secure' => true`, odśwież stronę kilka razy i zapisz, co dzieje się z licznikiem na `http://localhost` (w Chrome i w Firefoksie, jeśli masz). Wyłącz opcję i dopisz komentarz, kiedy należy ją włączyć.

3 Koszyk w sesji (sklep.php)

Wszystkie zmiany koszyka wykonuj metodą POST (formularz z ukrytym polem `id` i przyciskiem `name="akcja"`), a po obsłudze przekieruj na `sklep.php`.

- Wyświetl produkty z ceną i przyciskiem „Do koszyka” (akcja `dodaj`). Dodawaj tylko id istniejące w `$produkty`.
- Pod listą pokaż koszyk: produkt, ilość, wartość, przycisk „-1” (akcja `usun`, przy zerze pozycja znika), wiersz „Razem” z liczbą sztuk i kwotą oraz przycisk „Wyczyść koszyk” (akcja `wyczysc`, `unset`). Pusty koszyk: „Koszyk jest pusty.”.
- Test: 2 × Bazylia, 1 × Doniczka, -1 Bazylia, 1 × Pomidor. Zapisz kwotę. Otwórz stronę w oknie prywatnym (incognito) — co jest w koszyku i dlaczego?

4 Analiza w F12

- F12 → Aplikacja (Firefox: Dane) → Ciasteczka: zapisz w tabeli nazwę, wartość, datę wygaśnięcia, HttpOnly i SameSite dla `PHPSESSID`, `motyw`, `odwiedziny`.
- W konsoli F12 wpisz `document.cookie` i porównaj z tabelą. W zakładce Sieć znajdź odpowiedź z nagłówkiem `Set-Cookie`. Zrób zrzuty, spakuj folder.

W sprawozdaniu umieść

- ☐ tabela ciasteczek z F12 (nazwa, wartość, wygaśnięcie, HttpOnly, SameSite)
- ☐ wynik `document.cookie` i zrzut nagłówka `Set-Cookie` z zakładki Sieć
- ☐ wynik testu koszyka i obserwacja w oknie prywatnym

- ☐ obserwacja licznika z włączonym `secure` na `http://localhost`
- ☐ kod obsługi koszyka (akcje dodaj, usun, wyczysc)
- ☐ spakowany folder `nazwisko_z11`

Pytania kontrolne

1. Czym różni się przechowywanie danych w ciasteczku od przechowywania w sesji? Gdzie fizycznie leży koszyk?
2. Co dają atrybuty `HttpOnly`, `Secure` i `SameSite=Lax`?
3. Dlaczego `setcookie` i `session_start` muszą być wywołane przed wysłaniem jakiegokolwiek HTML?
4. Po co wywołuje się `session_regenerate_id(true)` po zalogowaniu użytkownika?

Kryteria oceny

Kryterium	Punkty
wspolne.php: parametry ciasteczka sesji, walidacja motywu, funkcja nagłówka z licznikiem koszyka	3
Licznik odwiedzin z walidacją i atrybutami, komunikat pierwszej i kolejnej wizyty	3
Motyw w ciasteczku (POST + PRG) na obu podstronach, usuwanie ciasteczek, test Secure	4
Koszyk w sesji: dodaj, -1, wyczyść, suma, pusty koszyk, PRG, kontrola id	5
Analiza F12 (tabela ciasteczek, document.cookie, Set-Cookie) i test okna prywatnego	3
Odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Rejestracja i logowanie

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie kompletnego mechanizmu kont użytkowników w PHP: rejestracja z bezpiecznym przechowywaniem haseł, logowanie z sesją, ochrona podstron i podział na role.

Po zajęciach potrafisz:

- zapisuje hasła wyłącznie jako skróty `password_hash()` i sprawdza je `password_verify()`
- waliduje dane rejestracji po stronie serwera i wykrywa zajęty login lub e-mail
- loguje użytkownika do sesji, zmieniając jej identyfikator `session_regenerate_id(true)`
- chroni podstrony funkcjami wymagającymi zalogowania lub roli admin (kod 403)
- poprawnie wylogowuje użytkownika, usuwając dane sesji i ciasteczko

Sprzęt i materiały

- komputer z Windows 11, XAMPP (Apache + MariaDB), przeglądarka z narzędziami F12
- plik `r2_12_portal.sql` i folder `r2_12_start` z folderu Pliki_do_zadan

Oprogramowanie

- **XAMPP** — Apache, PHP 8.2, MariaDB, phpMyAdmin
apachefriends.org/download.html
- **Visual Studio Code** — edycja plików PHP
code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 2.09 działanie sesji i funkcji `password_hash()` / `password_verify()` oraz wzorzec PRG z zadania 7.

Bezpieczeństwo

- Do testów używaj wymyślonych kont i haseł — nigdy swoich prawdziwych haseł z innych serwisów.
- Nie zapisuj haseł jawnym tekstem ani w bazie, ani w plikach, ani w sprawozdaniu (zrzut tabeli pokazuje tylko skróty).
- Serwer XAMPP służy tylko do pracy lokalnej — nie udostępniaj go w sieci.

Przebieg zadania

1 Przygotowanie projektu i bazy

Skopiuj folder `r2_12_start` do `C:\xampp\htdocs\portal`. Strony otwieraj przez `http://localhost/portal/`.

1. Zaimportuj `r2_12_portal.sql` w phpMyAdmin. Obejrzyj strukturę tabeli `uzytkownicy`: dlaczego kolumna `haslo` ma 255 znaków, a `login` i `email` mają indeks UNIQUE?
2. W `init.php` przed `session_start()` ustaw parametry ciasteczka sesji funkcją `session_set_cookie_params()` z tablicą: `lifetime 0`, `path '/'`, `httponly true`, `samesite 'Lax'`, `secure` zależne od tego, czy połączenie jest szyfrowane (`!empty($_SERVER['HTTPS'])`).

2 Rejestracja

Uzupełnij `rejestracja.php`. Komunikaty błędów zbieraj w tablicy `$bledy` — szablon już je wyświetla.

1. Walidacja: login pasuje do wyrażenia `/^[a-z0-9_]{3,20}$/`, e-mail przechodzi `filter_var(..., FILTER_VALIDATE_EMAIL)`, hasło ma co najmniej 8 znaków (`mb_strlen`), oba hasła są identyczne. Po błędzie formularz zachowuje login i e-mail, ale nie hasła.
2. Zapytaniem przygotowanym sprawdź, czy login lub e-mail nie są już zajęte — pokaż osobny komunikat dla każdego przypadku.
3. Gdy dane są poprawne, zapisz użytkownika z hasłem `password_hash($haslo, PASSWORD_DEFAULT)`, ustaw komunikat `flash('Konto założone...')` i przekieruj do `logowanie.php` (PRG). Załóż dwa konta testowe i obejrzyj kolumnę `haslo` w phpMyAdmin — co oznacza początek `$2y$10$`?

3 Logowanie i strefa dla zalogowanych

Dane zalogowanego trzymaj w sesji: `$_SESSION['uid']`, `['login']`, `['rola']`.

```
function zalogowany(): bool { ... }  
function wymagaj_logowania(): void { ... } // brak sesji → flash + Location: logowanie.php  
function wymagaj_rola(string $rola): void { ... } // inna rola → kod 403
```

1. W `logowanie.php` pobierz użytkownika po loginie i sprawdź hasło `password_verify()`. Przy błędzie pokaż jeden wspólny komunikat „Nieprawidłowy login lub hasło.” bez względu na to, czy pomyłono login, czy hasło.
2. Po poprawnym logowaniu wywołaj `session_regenerate_id(true)`, zapisz dane w sesji, uaktualnij kolumnę `ostatnie_logowanie` i przekieruj do `panel.php`. W F12 → Application (Aplikacja) → Cookies zanotuj wartość `PHPSESSID` przed i po zalogowaniu.
3. Dopisz w `init.php` trzy funkcje z ramki powyżej. Utwórz `panel.php` (dane zalogowanego z bazy w tabeli) chroniony przez `wymagaj_logowania()`. Sprawdź wejście na `panel.php` bez logowania.

4 Role, menu i wylogowanie

1. Utwórz `admin.php` z listą wszystkich użytkowników (bez kolumny hasła), chroniony przez `wymagaj_rola('admin')`. Wejdź na nią jako zwykły użytkownik — w F12 → Network sprawdź kod odpowiedzi.
2. Nadaj jednemu kontu rolę admin w phpMyAdmin (`UPDATE uzytkownicy SET rola = 'admin' WHERE login = '...';`). Czy zmiana działa od razu, czy dopiero po ponownym zalogowaniu? Dlaczego?
3. Zmień `menu.php`: gość widzi Rejestracja i Logowanie; zalogowany — Panel, swój login i przycisk Wyloguj; admin dodatkowo odnośnik Administracja.
4. Utwórz `wyloguj.php` obsługujący tylko POST (formularz z przyciskiem w menu): wyczyść `$_SESSION`, usuń ciasteczko sesji (`setcookie` z datą w przeszłości), wywołaj `session_destroy()` i przekieruj na stronę startową.
5. Wykonaj zrzuty: błędy rejestracji, tabela `uzytkownicy` w phpMyAdmin, panel, strona admina, odpowiedź 403 w F12. Spakuj folder `portal` jako `nazwisko_imie_zad12.zip`.

W sprawozdaniu umieść

- ☐ tabela wykonanych poleceń (numer kroku i punktu, np. 2.3) z kluczowymi fragmentami kodu
- ☐ zrzuty: błędy walidacji, tabela `uzytkownicy` ze skrótami haseł, panel, strona admina, kod 403 w F12
- ☐ wartości `PHPSESSID` przed i po zalogowaniu z komentarzem
- ☐ archiwum `nazwisko_imie_zad12.zip` z projektem
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego hasła nie szyfruje się ani nie zapisuje skrótem MD5, tylko używa `password_hash()`? Co zawiera zapisany skrót oprócz samego wyniku?
2. Po co zmieniać identyfikator sesji tuż po zalogowaniu?
3. Dlaczego przy nieudanym logowaniu podajemy jeden wspólny komunikat zamiast „nie ma takiego loginu”?
4. Czym różni się ukrycie odnośnika „Administracja” w menu od sprawdzenia roli w `admin.php`? Które jest zabezpieczeniem?

Kryteria oceny

Kryterium	Punkty
Rejestracja: walidacja 4 warunków, zachowanie pól, wykrycie zajętego loginu i e-maila	5
Hasła zapisane <code>password_hash()</code> , logowanie z <code>password_verify()</code> i wspólnym komunikatem	4
Sesja: parametry ciasteczka, <code>session_regenerate_id(true)</code> , panel chroniony przed gościem	4
Rola admin: strona admina z kodem 403 dla innych, menu zależne od stanu i roli	3
Wylogowanie POST usuwające sesję i ciasteczko; zrzuty i archiwum	2
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Upload plików, obsługa błędów i import CSV

Czas: 2 × 45 min

praca indywidualna

Cel: Bezpieczne przyjmowanie plików od użytkownika w PHP, czytelna obsługa i logowanie błędów oraz import danych z pliku CSV do bazy.

Po zajęciach potrafisz:

- buduje formularz wysyłania plików i odczytuje dane z `$_FILES` wraz z kodem błędu
- sprawdza rozmiar i rzeczywisty typ pliku (`fileinfo`), nadaje unikalną nazwę i zapisuje plik poza zasięgiem wykonywania skryptów
- tworzy miniatury biblioteką GD
- rozdziela wyświetlanie błędów (tryb deweloperski) od ich logowania do pliku
- importuje plik CSV do bazy z walidacją wierszy i raportem z importu

Sprzęt i materiały

- komputer z Windows 11, XAMPP, przeglądarka z narzędziami F12
- `r2_13_galeria.sql`, `r2_13_autorzy.csv`, foldery `r2_13_start` i `r2_13_testowe` z Pliki_do_zadan

Oprogramowanie

- **XAMPP** — PHP z rozszerzeniami `fileinfo` i `gd` (sprawdź w `phpinfo()`)
[apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code**
code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 2.10 budowę tablicy `$_FILES` i funkcję `move_uploaded_file()`.

Bezpieczeństwo

- Nigdy nie ufaj nazwie ani typowi pliku podanym przez przeglądarkę — sprawdzaj zawartość i nadawaj własną nazwę.
- Do testów używaj tylko plików z folderu `r2_13_testowe` lub własnych zdjęć bez danych osobowych (zdjęcia osób publikuje się tylko za ich zgodą).
- Plik logu może zawierać szczegóły techniczne — nie może być dostępny z przeglądarki.

Przebieg zadania

1 Projekt i formularz wysyłania

Skopiuj `r2_13_start` do `C:\xampp\htdocs\galeria`, zaimportuj `r2_13_galeria.sql`. W `phpinfo()` sprawdź, czy są sekcje **fileinfo** i **gd** — jeśli nie, w `C:\xampp\php\php.ini` usuń średnik przed `extension=fileinfo` / `extension=gd` i zrestartuj Apache. Zanotuj wartości `upload_max_filesize` i `post_max_size`.

1. W `dodaj.php` uzupełnij formularz: `enctype="multipart/form-data"`, ukryte pole `MAX_FILE_SIZE` (wartość z `MAX_ROZMIAR`) przed polem pliku, atrybuty `accept` i `required` w polu `zdjecie`.
2. Na początku obsługi POST wypisz tymczasowo `var_dump($_FILES)` i wyślij `gory.jpg`. Opisz w sprawozdaniu znaczenie pól `name`, `type`, `tmp_name`, `error`, `size`. Usuń `var_dump`.

2 Bezpieczne przyjęcie pliku

Błędy zbieraj w tablicy `$bledy`. Kolejność kontroli: tytuł → kod błędu → rozmiar → typ.

```
$typ = (new finfo(FILEINFO_MIME_TYPE))->file($plik['tmp_name']);  
$nazwa = bin2hex(random_bytes(16)) . '.' . DOZWOLONE_TYPY[$typ];
```

1. Napisz funkcję `komunikat_uploadu(int $kod): string` (instrukcja `match`), która zamienia kody `UPLOAD_ERR_INI_SIZE`, `UPLOAD_ERR_FORM_SIZE`, `UPLOAD_ERR_PARTIAL`, `UPLOAD_ERR_NO_FILE` i pozostałe na polskie komunikaty.
2. Sprawdź rozmiar (`MAX_ROZMIAR`) i typ funkcją `finfo` (ramka) — dozwolone tylko typy z `DOZWOLONE_TYPY`. Rozszerzenie zapisanego pliku bierz z tej tablicy, nie z nazwy od użytkownika.
3. Zapisz plik `move_uploaded_file()` pod losową nazwą w `uploads/`, a dane (tytuł, autor, nazwa, oryginalna nazwa, typ, rozmiar) w tabeli `zdjecia` zapytaniem przygotowanym. Po sukcesie przekieruj do `index.php?dodano=1`.
4. Napisz funkcję `utworz_miniature()`, która biblioteką GD (`imagecreatefromjpeg/png/webp`, `imagescale` do szerokości 400 px, `imagejpeg`) zapisze miniaturę w `uploads/min/`. Gdy GD nie ma (`function_exists('imagecreatetruecolor')`), funkcja zwraca `null`, a galeria pokazuje oryginał.
5. Przetestuj wszystkie pliki z `r2_13_testowe`. Oczekiwane: 3 zdjęcia przyjęte, `duzy_szum.png` — za duży, `notatka.jpg` i `prostokat.gif` — niedozwolony typ.

3 Galeria i ochrona katalogu

1. W `index.php` pobierz zdjęcia z imieniem i nazwiskiem autora (`LEFT JOIN`, najnowsze pierwsze) i wyświetl kafelki według wzoru w komentarzu: miniatura (`loading="lazy"`, `alt` = tytuł) z odnośnikiem do oryginału, tytuł, autor lub „autor nieznany”, rozmiar w KB. Wszystkie dane wypisuj przez `e()`.
2. Utwórz `uploads/.htaccess`, który wyłącza listowanie katalogu (`Options -Indexes`) i blokuje dostęp do plików z rozszerzeniami skryptów PHP (`<FilesMatch> + Require all denied`). Sprawdź, że `http://localhost/galeria/uploads/` zwraca 403.

4 Obsługa i logowanie błędów

Uzupełnij `config.php` w miejscu oznaczonym TODO.

1. Ustaw `error_reporting(E_ALL)`, `display_errors` zależne od `TRYB_DEV`, `log_errors` i `error_log = PLIK_LOGU`. Utwórz `logs/.htaccess` z `Require all denied`.
2. Napisz funkcję `loguj(string $poziom, string $tekst)` dopisującą do logu wiersz `[data czas] POZIOM: tekst` (`error_log($linia, 3, PLIK_LOGU)`). Loguj dodanie zdjęcia i każdy odrzucony plik.
3. Dodaj `set_exception_handler()`: wpis do logu + kod 500 + w trybie DEV szczegóły, w produkcji tylko „Przepraszamy, wystąpił błąd”. Test: zmień w `db.php` nazwę bazy na błędną, otwórz galerię przy `TRYB_DEV = true` i `false`, obejrzyj log, przywróć poprawną nazwę.

5 Import autorów z CSV

Otwórz `r2_13_autorzy.csv` w Notatniku. Formularz w `import.php` przyjmuje plik CSV (`$_FILES['csv']`). Czytaj go wywołaniem `fgetcsv($f, null, ';', '"', '')`.

1. Sprawdź nagłówek (`imie;nazwisko;klasa`), usuwając z niego znak BOM (`preg_replace('/^\xEF\xBB\xBF/', '', ...)`).
2. Każdy wiersz zvaliduj: 3 kolumny, niepuste imię i nazwisko, klasa zgodna z `/^[1-5]T[IP]$/`. Poprawne wstaw przez `INSERT IGNORE` (w tabeli jest UNIQUE na imię, nazwisko i klasę), całość w transakcji.
3. Pokaż raport: dodane, pominięte (już istnieją — `rowCount() = 0`), błędne z numerem wiersza i powodem; zapisz go też w logu. Zimportuj plik dwa razy i porównaj raporty. Pod formularzem wyświetl tabelę autorów.
4. Zrzuty: komunikaty błędów uploadu, galeria, 403 dla `uploads/`, oba raporty importu, fragment `app.log`. Spakuj projekt (bez folderu `uploads`) jako `nazwisko_imie_zad13.zip`.

W sprawozdaniu umieść

- ☐ tabela wykonanych poleceń (krok.punkt) z kluczowymi fragmentami kodu
- ☐ opis pól `$_FILES` i wartości `upload_max_filesize` / `post_max_size`
- ☐ zrzuty: błędy uploadu, galeria, 403 dla katalogu, raporty importu, fragment logu
- ☐ archiwum `nazwisko_imie_zad13.zip`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego nie można polegać na `$_FILES['zdjecie']['type']` ani na rozszerzeniu pliku?
2. Po co serwer nadaje plikom własne, losowe nazwy?
3. Dlaczego na serwerze produkcyjnym `display_errors` musi być wyłączone, a błędy i tak powinny być zapisywane?
4. Co się stanie z `$_POST` i `$_FILES`, gdy wysłany plik przekroczy `post_max_size`? Jak to wykryć?
5. Dlaczego import CSV warto wykonać w transakcji?

Kryteria oceny

Kryterium	Punkty
Formularz z enctype i MAX_FILE_SIZE, opis <code>\$_FILES</code>	2
Kontrola kodu błędu, rozmiaru i typu (finfo); 3 pliki odrzucone z właściwym komunikatem	4
Losowa nazwa, zapis do bazy, miniatura GD	3
Galeria z escapowaniem, .htaccess w <code>uploads/</code> (403)	2
Konfiguracja błędów, funkcja <code>loguj()</code> , obsługa wyjątków w DEV i produkcji	3
Import CSV: BOM, walidacja, INSERT IGNORE w transakcji, raport 8/1/3	4
Zrzuty, archiwum i odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Bezpieczeństwo — audyt i poprawa aplikacji

Czas: 2 × 45 min

praca indywidualna lub w parach (audyt wspólnie, poprawki indywidualnie)

Cel: Przeprowadzenie przeglądu bezpieczeństwa małej aplikacji PHP według listy kontrolnej i poprawienie kodu tak, aby stosował dobre praktyki obronne.

Po zajęciach potrafisz:

- wyszukuje w kodzie miejsca niezgodne z listą kontrolną i dokumentuje je w tabeli audytu
- zastępuje doklejanie danych do SQL zapytaniami przygotowanymi `mysqli_prepare`
- escapuje dane na wyjściu i wdraża tokeny CSRF w formularzach zmieniających dane
- sprawdza uprawnienia po stronie serwera (właściciel lub admin) i poprawnie kończy skrypt po przekierowaniu
- ukrywa szczegóły błędów przed użytkownikiem i ustawia nagłówki bezpieczeństwa z Content-Security-Policy

Sprzęt i materiały

- komputer z Windows 11, XAMPP, przeglądarka z narzędziami F12
- `r2_14_tablica.sql` i folder `r2_14_start` z Pliki_do_zadan

Oprogramowanie

- **XAMPP**
apachefriends.org/download.html
- **Visual Studio Code** — wyszukiwanie w plikach: Ctrl+Shift+F
code.visualstudio.com
- **OWASP Top 10** — opisy kategorii ryzyk i zalecane zabezpieczenia
owasp.org/Top10

Przygotowanie przed zajęciami

- Przejrzyj wykład 2.11 (zasady obrony) i własny kod z zadania 12.

Bezpieczeństwo

- Pracujesz wyłącznie na własnej kopii aplikacji na `localhost`. Nie testujesz żadnych cudzych stron ani serwerów — to niezgodne z prawem i regulaminem szkoły.
- Celem zadania jest poprawa kodu. Sprawdzasz działanie zwykłymi danymi (np. tytuł z apostrofem lub znakami < >), nie szukasz sposobów obejścia zabezpieczeń.
- Przed zmianami zrób kopię folderu `tablica_przed` — posłuży do porównania.

Przebieg zadania

1 Uruchomienie i audyt

Skopiuj `r2_14_start` do `htdocs\tablica` (i kopię do `htdocs\tablica_przed`), zaimportuj `r2_14_tablica.sql`. Konta: admin / Admin2026!, anna / Anna2026!, piotr / Piotr2026!.

LISTA KONTROLNA

- [1] Dane od użytkownika trafiają do SQL wyłącznie przez zapytania przygotowane.
- [2] Każda dana wypisywana w HTML przechodzi przez `htmlspecialchars` (funkcja `e()`).
- [3] Formularze POST zmieniające dane mają token CSRF; zmiany danych tylko metodą POST.
- [4] Serwer sprawdza zalogowanie i uprawnienia (właściciel/admin); po `header('Location')` jest `exit`.
- [5] Sesja: ciasteczko `HttpOnly` + `SameSite`, `session_regenerate_id(true)` po zalogowaniu.
- [6] Użytkownik nie widzi treści błędów ani zapytań SQL; błędy trafiają do logu.
- [7] Nagłówki: `Content-Security-Policy`, `X-Content-Type-Options`, `Referrer-Policy`, `X-Frame-Options`.

1. Zaloguj się jako anna i przejrzyj aplikację: lista, wyszukiwanie, dodawanie, usuwanie. Zauważ, przy których ogłoszeniach widać „Usuń”.
2. Przeczytaj wszystkie pliki PHP i wypełnij tabelę audytu w sprawozdaniu: plik, nr linii, punkt listy [1]–[7], opis problemu, planowana poprawka. Znajdź co najmniej 12 niezgodności.
3. W F12 → Network obejrzyj nagłówki odpowiedzi `index.php` i zapisz, których z punktu [7] brakuje.

2 Zapytania przygotowane i escapowanie [1] [2]

Utwórz plik `bezpieczenstwo.php` dołączany w `db.php`; w nim funkcja `e()` i pozostałe funkcje z kolejnych kroków.

1. Przepisz wszystkie zapytania z danymi od użytkownika na `mysqli_prepare` + `mysqli_stmt_bind_param` + `mysqli_stmt_get_result` (wyszukiwanie z `LIKE ?` i wzorcem `'%' . $q . '%'`, logowanie, dodawanie, usuwanie). Identyfikator rzutuj/waliduj jako liczbę.
2. Każdą daną z bazy, sesji i formularza wypisuj przez `e()`; treść ogłoszenia dodatkowo `nl2br(e(...))`. Sprawdź: ogłoszenie z tytułem `Test <b>` wyświetla się dosłownie, wyszukiwanie słowa z apostrofem (np. O'Neil) nie powoduje błędu.

3 CSRF, uprawnienia i sesja [3] [4] [5]

```
function csrf_token(): string // losowy token w $_SESSION['csrf'] (random_bytes)
function csrf_pole(): string  // <input type="hidden" name="csrf" value="...">
function sprawdz_csrf(): void  // hash_equals(...), inaczej kod 403 i exit
```

1. Napisz trzy funkcje z ramki i dodaj token do formularzy: logowanie, dodawanie, usuwanie, wylogowanie. W każdej obsłudze POST najpierw wywołaj `sprawdz_csrf()`.
2. Zamień odnośnik „Usuń” na formularz POST z ukrytym `id` i tokenem. `usun.php`: dla GET kod 405, brak ogłoszenia — 404, cudze ogłoszenie — 403 (usuwać może autor albo admin — funkcja `moze_usunac()`); przycisk widzą tylko uprawnieni.
3. Popraw `dodaj.php`: po `header('Location: ...')` dodaj `exit`, waliduj długość tytułu (1–100) i treści (1–2000). Wylogowanie zrób formularzem POST.
4. Ustaw parametry ciasteczka sesji (`HttpOnly`, `SameSite=Lax`) i dodaj `session_regenerate_id(true)` po zalogowaniu. Komunikat błędnego logowania nie powtarza wpisanego loginu.

4 Błędy i nagłówki [6] [7]

1. W `db.php` wyłącz `display_errors`, włącz logowanie do `logs/php_errors.log` (z `logs/.htaccess: Require all denied`), włącz `mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT)` i dodaj `set_exception_handler` z kodem 500 i ogólnym komunikatem. Usuń wszystkie `or die(mysqli_error(...))`. Sprawdź działanie, zmieniając chwilowo nazwę bazy.
2. Na początku `bezpieczenstwo.php` wyślij nagłówki z punktu [7]. CSP: `default-src 'self'; script-src 'self'; style-src 'self'; img-src 'self' data:; object-src 'none'; base-uri 'self'; form-action 'self'; frame-ancestors 'none'`.
3. Z polityką CSP atrybut `onclick` przestanie działać — sprawdź komunikat w konsoli F12. Przenieś potwierdzenie usunięcia do pliku `skrypt.js` (nasłuch `submit` na formularzach `.usun`, dołączony z `defer`).
4. Uzupełnij kolumnę „Poprawiono” w tabeli audytu, wykonaj zrzuty (nagłówki w F12, 403 przy cudzym ogłoszeniu, 405 dla GET, strona błędu 500) i spakuj projekt jako `nazwisko_imie_zad14.zip`.

W sprawozdaniu umieść

- ☐ tabela audytu (co najmniej 12 pozycji) z kolumną „Poprawiono”
- ☐ fragmenty kodu przed i po dla punktów [1], [2], [3] i [4]
- ☐ zrzuty: nagłówki odpowiedzi przed i po, 403 / 405, strona błędu 500, konsola F12 z komunikatem CSP
- ☐ archiwum `nazwisko_imie_zad14.zip`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego zapytanie przygotowane chroni bazę skuteczniej niż ręczne „czyszczenie” danych?
2. Co sprawdza token CSRF i dlaczego musi być losowy i powiązany z sesją?
3. Dlaczego ukrycie przycisku „Usuń” nie wystarcza i serwer musi sprawdzić uprawnienia?
4. Jakie informacje mogłyby zdradzić komunikaty `mysqli_error()` pokazywane użytkownikowi?
5. Do czego służy nagłówek Content-Security-Policy i dlaczego wymaga przeniesienia skryptów inline do plików?

Kryteria oceny

Kryterium	Punkty
Tabela audytu: min. 12 trafnych niezgodności z plikiem, linią i punktem listy	4
[1] Wszystkie zapytania przygotowane; [2] pełne escapowanie wyjścia	4
[3] Token CSRF w 4 formularzach, usuwanie i wylogowanie tylko POST	3
[4] Kontrola uprawnień (403/404/405), <code>exit</code> po przekierowaniu, walidacja długości; [5] sesja	3
[6] Ukryte błędy + log; [7] nagłówki i CSP, skrypt w pliku zewnętrznym	3
Zrzuty, archiwum, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

PHP obiektowy — sklep w klasach

Czas: 2 × 45 min

praca indywidualna

Cel: Zaprojektowanie i napisanie kilku współpracujących klas PHP z hermetyzacją, interfejsem i przestrzeniami nazw oraz sprawdzenie ich działania skryptem testów.

Po zajęciach potrafisz:

- tworzy klasy z konstruktorem z promocją właściwości i walidacją danych (wyjątki)
- ukrywa stan obiektu (`private`, `readonly`) i udostępnia go przez metody
- definiuje interfejs i kilka jego implementacji, korzystając z polimorfizmu
- porządkuje klasy w przestrzeniach nazw i plikach zgodnie z PSR-4
- uruchamia skrypt testów w konsoli i interpretuje jego wyniki

Sprzęt i materiały

- komputer z Windows 11 i XAMPP (PHP 8.2 lub nowszy)
- folder `r2_15_start` z Pliki_do_zadan

Oprogramowanie

- **XAMPP** — PHP w konsoli:
`C:\xampp\php\php.exe` (przycisk Shell w panelu XAMPP)
[apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code** — terminal: Ctrl+`;
rozszerzenie PHP Intelephense
podpowiada metody klas
code.visualstudio.com

Przygotowanie przed zajęciami

- Przeglądnij wykład 2.12: klasy, konstruktor z promocją właściwości, interfejsy, przestrzenie nazw.

Bezpieczeństwo

- Nie modyfikuj testów w `test.php`, aby „przeszły” — popraw klasy. Możesz dopisywać własne testy na końcu pliku.

Przebieg zadania

1 Klasa Produkt

Skopiuj `r2_15_start` do `htdocs\sklep_oop`. W terminalu VS Code uruchamiaj `php test.php` (jeśli polecenie `php` nie działa, użyj `C:\xampp\php\php.exe test.php`). Każdy plik zaczyna od `<?php`, `declare(strict_types=1);` i deklaracji `namespace`.

```
namespace Sklep;                                // plik src/Produkt.php
class Produkt
{
    public const STAWKI_VAT = [0, 5, 8, 23];
    __construct(string $nazwa, float $cenaNetto, int $stawkaVat = 23)
    {
        getNazwa(): string    getCenaNetto(): float    getStawkaVat(): int
        cenaBrutto(): float    // zaokrąglona do 2 miejsc
        obnizCene(float $procent): void // procent z przedziału (0, 100)
    }
}
```

1. Uruchom `php test.php` przed napisaniem kodu — wszystkie sekcje mają wynik `[--]`.
2. Napisz klasę `Produkt` według ramki. Właściwości zadeklaruj w konstruktorze (promocja) jako `private`. Pusta nazwa, ujemna cena lub stawka spoza `STAWKI_VAT` mają zgłaszać `InvalidArgumentException`.
3. Uruchom testy — sekcja Produkt ma 6 × `[OK]`. Sprawdź, co się stanie, gdy w `test.php` dopiszesz na chwilę `echo $mysz->cenaNetto;` (zanotuj komunikat i usuń wiersz).

2 Klasa Koszyk

Koszyk przechowuje pozycje w prywatnej tablicy: klucz = nazwa produktu, wartość = `['produkt' => Produkt, 'ilosc' => int]`.

```
class Koszyk                                // src/Koszyk.php
{
    dodaj(Produkt $produkt, int $ilosc = 1): void // ten sam produkt → zwiększa ilość
    usun(string $nazwa): void
    zmienIlosc(string $nazwa, int $ilosc): void // 0 lub mniej → usuwa pozycję
    czyPusty(): bool    liczbaSztuk(): int    wartoscBrutto(): float
    getPozycje(): array
}
```

1. Napisz klasę `Koszyk` według ramki. Ilość mniejsza niż 1 w `dodaj()` i zmiana ilości produktu, którego nie ma w koszyku, zgłaszają `InvalidArgumentException`.
2. Uruchom testy — sekcja Koszyk ma 6 × `[OK]`. W sprawozdaniu wyjaśnij, dlaczego `getPozycje()` zwracająca tablicę nie pozwala zmienić zawartości koszyka z zewnątrz.

3 Interfejs SposobDostawy

Przestrzeń nazw `Sklep\Dostawa`, pliki w `src/Dostawa/`.

```
interface SposobDostawy
{
    nazwa(): string
    koszt(float $wartoscKoszyka): float
}

Kurier          — „Kurier”, 19.99 zł; od 300 zł gratis
Paczkomat       — konstruktor z kodem punktu, nazwa „Paczkomat KOD”, 12.99 zł; od 200 zł gratis
OdbiorOsobisty — „Odbiór osobisty w sklepie”, 0 zł
```

1. Napisz interfejs `SposobDostawy` i trzy klasy, które go implementują (`implements`).
2. Uruchom testy — sekcja Dostawa ma 4 × `[OK]`. Dopisz własną klasę `PocztaPolska` (koszt 15.50 zł, bez darmowej dostawy) i własny test jej kosztu.

4 Klasa Zamowienie i podsumowanie

```
class Zamowienie // src/Zamowienie.php, use
{
    Sklep\Dostawa\SposobDostawy;
    private static int $licznik = 0;
    public readonly string $numer; // ZAM/2026/0001, ZAM/2026/0002, ...
    __construct(Koszyk $koszyk, SposobDostawy $dostawa, string $email)
    {
        $this->kosztDostawy();
        $this->razem();
        $this->podsumowanie();
    }
}
```

1. Napisz klasę `Zamowienie`. Konstruktor odrzuca pusty koszyk i niepoprawny e-mail (`filter_var`), zwiększa statyczny licznik i nadaje numer `sprintf('ZAM/%s/%04d', date('Y'), ...)`.
2. `kosztDostawy()` pyta obiekt dostawy o koszt dla wartości koszyka — klasa `Zamowienie` nie wie, czy to kurier, czy paczkomat. `podsumowanie()` zwraca wielowierszowy tekst: numer i e-mail, pozycje (nazwa, ilość, cena), dostawa z kosztem, RAZEM; kwoty formatuj `number_format($x, 2, ',', ' ')`.
3. Uruchom testy — wynik ma być 23 OK, 0 błędów (plus Twoje testy). Zrób zrzut terminala.
4. Utwórz `sklep.php` — stronę HTML, która tworzy 3 produkty, koszyk i zamówienie z odbiorem osobistym, a podsumowanie wyświetla w `<pre>` (pamiętaj o `htmlspecialchars`). Otwórz ją przez `http://localhost/sklep_oop/sklep.php`. Spakuj projekt jako `nazwisko_imie_zad15.zip` — będzie potrzebny w zadaniu 16.

W sprawozdaniu umieść

- ☐ tabela wykonanych poleceń (krok.punkt) z najważniejszymi fragmentami kodu
- ☐ komunikat przy próbie odczytu prywatnej właściwości
- ☐ zrzut terminala z wynikiem testów i zrzut strony `sklep.php`
- ☐ diagram klas (prostokąty z nazwami klas i metod; strzałka „implementuje” od klas dostawy do interfejsu) — może być odręczny lub w draw.io
- ☐ archiwum `nazwisko_imie_zad15.zip` i odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Co daje hermetyzacja? Podaj przykład błędu, przed którym chroni prywatna `cenaNetto` z walidacją.
2. Dlaczego `Zamowienie` przyjmuje w konstruktorze typ `SposobDostawy`, a nie `Kurier`? Co trzeba zmienić w `Zamowienie`, gdy dochodzi nowy sposób dostawy?
3. Czym różni się właściwość statyczna `$licznik` od zwykłej właściwości obiektu?
4. Po co przestrzenie nazw i jak nazwa klasy `Sklep\Dostawa\Kurier` odpowiada ścieżce pliku?

Kryteria oceny

Kryterium	Punkty
Produkt: promocja właściwości, walidacja z wyjątkami, cena brutto (6 testów OK)	4
Koszyk: łączenie tych samych produktów, zmiana ilości, wartość (6 testów OK)	4
Interfejs i 3 implementacje + własna klasa <code>PocztaPolska</code> z testem	3
Zamowienie: walidacja, numeracja statyczna, readonly, podsumowanie (wszystkie testy OK)	4
Strona <code>sklep.php</code> , diagram klas, zrzuty, archiwum	3
Odpowiedzi na pytania kontrolne	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Composer, autoloading PSR-4 i biblioteka phpdotenv

Czas: 2 × 45 min

praca indywidualna

Cel: Zarządzanie zależnościami projektu PHP Composerem: autoloading klas zgodny z PSR-4, instalacja biblioteki z Packagist i przeniesienie konfiguracji bazy do pliku `.env` poza repozytorium.

Po zajęciach potrafisz:

- tworzy plik `composer.json` poleceniem `composer init` i rozumie jego sekcje
- konfiguruje autoloading PSR-4 i zastępuje nim ręczne `require`
- instaluje bibliotekę, odczytuje ograniczenia wersji (`^5.7`) i rolę pliku `composer.lock`
- przechowuje dane dostępowe w `.env` (phpdotenv) z walidacją wymaganych zmiennych
- przygotowuje `.gitignore` tak, by `vendor/` i `.env` nie trafiły do repozytorium

Sprzęt i materiały

- komputer z Windows 11, XAMPP, Composer, Git, dostęp do internetu (Packagist)
- projekt z zadania 15 (lub folder `r2_16_start`) i plik `r2_16_sklep_oop.sql`

Oprogramowanie

- **Composer** — instalator Composer-Setup.exe; jako PHP wskaż `C:\xampp\php\php.exe`
getcomposer.org/download
- **Packagist** — strona biblioteki phpdotenv
packagist.org/packages/vlucas/phpdotenv
- **Git**
git-scm.com/downloads
- **XAMPP i Visual Studio Code** — VS Code:
<https://code.visualstudio.com/apachefriends.org/download.html>

Przygotowanie przed zajęciami

- Sprawdź, czy na komputerze działa `composer --version`; jeśli nie, zainstaluj Composera (wymaga uprawnień — zgłoś nauczycielowi).

Bezpieczeństwo

- Plik `.env` zawiera dane dostępowe — nigdy nie wysyłaj go do repozytorium ani nie wklejaj do sprawozdania z prawdziwymi hasłami.
- Instaluj tylko biblioteki z wiarygodnych źródeł (Packagist, aktywne repozytorium, wiele pobrań); przed wdrożeniem uruchamiaj `composer audit`.

Przebieg zadania

1 composer init

Skopiuj projekt z zadania 15 do `htdocs\sklep_composer`. Polecenia wpisuj w terminalu VS Code w folderze projektu.

1. Uruchom `composer --version` i `php --version`; zanotuj wersje.
2. Uruchom `composer init`. Nazwa pakietu: `nazwisko/sklep-oop`, typ `project`, opis własny. Na pytania o interaktywne dodanie zależności odpowiedz `no`, na pytanie o mapowanie PSR-4 — `n` (dodasz je ręcznie). Obejrzyj powstały `composer.json`.

2 Autoload PSR-4

Zasada PSR-4: prefiks przestrzeni nazw odpowiada katalogowi, a reszta nazwy klasy — ścieżce pliku.

```
"autoload": {  
    "psr-4": {  
        "Sklep\\": "src/"  
    }  
}
```

1. Dopisz do `composer.json` sekcję z ramki i wykonaj `composer dump-autoload`. Obejrzyj folder `vendor/composer` — w którym pliku jest zapisane mapowanie `Sklep\\` → `src/`?
2. W `test.php` zastąp wszystkie `require_once` jednym `require __DIR__ . '/vendor/autoload.php'`; i uruchom testy — wynik ma być taki sam jak w zadaniu 15.
3. Dodaj nową klasę (np. `PocztaPolska`, jeśli jej nie masz) i sprawdź, że działa bez zmian w `test.php` poza testem. Kiedy po dodaniu klasy trzeba uruchomić `composer dump-autoload`, a kiedy nie?

3 Biblioteka phpdotenv i konfiguracja w .env

Zaimportuj `r2_16_sklep_oop.sql`.

```
# .env.example (wzór – trafia do repozytorium)  
APP_ENV=dev  
DB_HOST=localhost  
DB_NAME=sklep_oop  
DB_USER=root  
DB_PASS=
```

1. Zainstaluj bibliotekę: `composer require vlucas/phpdotenv`. Zanotuj, jakie ograniczenie wersji dopisał Composer w `require` i jakie dodatkowe pakiety zainstalował (`composer show`, `composer show --tree`).
2. Utwórz `.env.example` (ramka) i jego kopię `.env`. Utwórz `config/bootstrap.php`, który łączy autoload, wczytuje `.env` (`Dotenv\Dotenv::createImmutable(...)->load()`), wymaga niepustych `DB_HOST`, `DB_NAME`, `DB_USER`, istnienia `DB_PASS` i wartości `APP_ENV` z listy `dev / prod` (`required()`, `notEmpty()`, `allowedValues()`), a na tej podstawie ustawia `display_errors`.
3. Napisz klasę `Sklep\Baza` z metodą statyczną `połącz(): PDO`, która tworzy jedno połączenie z danymi z `$_ENV`, oraz `Sklep\ProduktRepozytorium::wszystkie()` zwracającą tablicę obiektów `Produkt` z tabeli `produkty`.
4. Utwórz `public/index.php`: tabela produktów z bazy (netto, VAT, brutto) i przykładowe zamówienie z dwóch produktów. Otwórz `http://localhost/sklep_composer/public/`.
5. Sprawdź walidację: usuń z `.env` wiersz `DB_USER`, a potem zmień nazwę pliku `.env` — zanotuj komunikaty wyjątków. Przywróć plik.

4 Wersje, lock i repozytorium

1. Wyjaśnij w sprawozdaniu zapis `^5.7` (jakie wersje dopuszcza) i porównaj z `~5.7` oraz `5.7.*`. Odczytaj w `composer.lock` dokładną zainstalowaną wersję phpdotenv.
2. Utwórz `.gitignore` z wpisami `/vendor/` i `.env`. Wykonaj `git init`, `git add .`, `git status` — sprawdź, że na liście nie ma `vendor/` ani `.env`, ale są `composer.json`, `composer.lock` i `.env.example`. Zatwierdź zmiany (`git commit`).
3. Test odtworzenia: skopiuj repozytorium do nowego folderu poleceniem `git clone`, uruchom tam `composer install`, utwórz `.env` z `.env.example` i sprawdź działanie strony. Uruchom `composer audit` i `composer outdated`. Spakuj projekt bez `vendor/` jako `nazwisko_imie_zad16.zip`.

W sprawozdaniu umieść

- ☐ tabela wykonanych poleceń (krok.punkt) z poleceniami i fragmentami kodu
- ☐ plik `composer.json` i wynik `composer show --tree`
- ☐ komunikaty phpdotenv przy braku zmiennej i braku pliku `.env`
- ☐ zrzuty: wynik testów z autoloadem, strona `public/`, `git status` po dodaniu `.gitignore`
- ☐ archiwum `nazwisko_imie_zad16.zip` (bez `vendor/`) i odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się `composer install` od `composer update`? Które polecenie wykonasz po pobraniu projektu kolegi i dlaczego?
2. Dlaczego `composer.lock` dodajemy do repozytorium, a `vendor/` nie?
3. Jakie korzyści daje trzymanie konfiguracji w `.env` zamiast w kodzie PHP?
4. Co oznacza ograniczenie wersji `^5.7` i dlaczego jest bezpieczniejsze niż `*`?

Kryteria oceny

Kryterium	Punkty
<code>composer.json</code> z init, autoloader PSR-4, testy działające przez <code>vendor/autoload.php</code>	5
Instalacja phpdotenv, <code>.env</code> / <code>.env.example</code> , bootstrap z walidacją zmiennych	4
Klasy <code>Baza</code> i <code>ProduktRepozytorium</code> , strona z produktami z bazy	4
Wersje i lock opisane poprawnie; <code>.gitignore</code> , commit, odtworzenie przez <code>composer install</code>	4
Zrzuty, archiwum, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Mini-MVC: front controller i routing

Czas: 2 × 45 min

praca indywidualna

Cel: Przebudowa prostej aplikacji CRUD na architekturę MVC z jednym punktem wejścia, routerem i przyjaznymi adresami URL.

Po zajęciach potrafisz:

- wyjaśnia rolę modelu, widoku i kontrolera oraz front controllera
- kieruje wszystkie żądania do `index.php` regułą `mod_rewrite` w `.htaccess`
- pisze router dopasowujący metodę HTTP i wzorzec adresu z parametrem `{id}`
- przenosi zapytania SQL do klasy modelu, a HTML do widoków ze wspólnym układem
- odróżnia adresy zasobów (`/filmy/5`) od plików skryptów (`edytuj.php?id=5`)

Sprzęt i materiały

- komputer z Windows 11, XAMPP (Apache z `mod_rewrite`), Composer
- aplikacja z zadania 9 albo folder `r2_17_start` i plik `r2_17_katalog.sql`

Oprogramowanie

- **XAMPP** — `mod_rewrite` i `AllowOverride All` dla `htdocs` są włączone w domyślnej konfiguracji — sprawdź w `httpd.conf`, jeśli adresy nie działają
apachefriends.org/download.html
- **Composer** — tylko autoloader (bez pakietów)
getcomposer.org/download
- **Visual Studio Code**
code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie autoloader PSR-4 z zadania 16 i działanie aplikacji z zadania 9.

Bezpieczeństwo

- Pliki konfiguracji, modele i widoki umieść poza folderem `public/` — z przeglądarki dostępny jest tylko `public/`.
- Usuwanie wykonuj wyłącznie metodą POST; zachowaj zapytania przygotowane i escapowanie z zadania 9.

Przebieg zadania

1 Analiza i struktura projektu

Zaimportuj `r2_17_katalog.sql`, skopiuj `r2_17_start` (lub zadanie 9) do `htdocs\katalog_stary` i sprawdź działanie. Nowy projekt utwórz w `htdocs\katalog_mvc`.

```
katalog_mvc/  
  app/Controllers/FilmController.php  app/Models/Film.php  
  app/Router.php  app/View.php  app/helpers.php  
  views/layout.php  views/404.php  views/filmy/{lista,pokaz,formularz}.php  
  public/index.php  public/.htaccess  public/style.css  
  config.php  composer.json
```

1. W tabeli w sprawozdaniu przypisz każdy fragment starej aplikacji (zapytania SQL, walidacja, HTML, obsługa POST, przekierowania) do warstwy: model, widok, kontroler.
2. Utwórz strukturę z ramki. W `composer.json` ustaw autoload `"App\\": "app/"` oraz `"files": ["app/helpers.php"]`, uruchom `composer dump-autoload`. W `config.php` zwróć tablicę z danymi połączenia (`return [...]`).

2 Front controller, .htaccess i router

Docelowe trasy aplikacji:

GET /	→ FilmController::index	lista
GET /filmy/dodaj	→ create	formularz
POST /filmy	→ store	zapis nowego
GET /filmy/{id}	→ show	szczegóły
GET /filmy/{id}/edytuj	→ edit	formularz edycji
POST /filmy/{id}	→ update	zapis zmian
POST /filmy/{id}/usun	→ destroy	usunięcie

1. Utwórz `public/.htaccess`: `RewriteEngine On`, dwa warunki `RewriteCond %{REQUEST_FILENAME} !-f` i `!-d`, reguła `RewriteRule ^ index.php [QSA,L]`. Wyjaśnij, po co są warunki.
2. W `app/helpers.php` napisz `baza_url()` (katalog z `$_SERVER['SCRIPT_NAME']`), np. `/katalog_mvc/public`), `url($sciezka)`, `e()` i `przekieruj($sciezka, $flash)` (nagłówek Location + `exit`).
3. Napisz klasę `App\Router` z metodami `get()`, `post()` i `dispatch($metoda, $uri)`: usuń z adresu część bazową i zapytanie (`parse_url`), zamień `{id}` we wzorcu na `(\d+)`, przy dopasowaniu wywołaj akcję z parametrami jako `int`; brak dopasowania → kod 404 i widok `404`.
4. W `public/index.php` (front controller) dołącz autoload, uruchom sesję, utwórz PDO z `config.php`, model i kontroler, zarejestruj trasy z ramki i wywołaj `dispatch()`.

3 Model, widoki i kontroler

1. Klasa `App\Models\Film` dostaje PDO w konstruktorze i ma metody `wszystkie()`, `znajdz(int $id): ?array`, `dodaj(array): int` (zwraca `lastInsertId`), `zmien()`, `usun()` oraz statyczną `waliduj(array $d): array` zwracającą `[dane, błędy]` (przenieś walidację ze starej aplikacji). Gatunki jako stała klasy.
2. Klasa `App\View` z metodą statyczną `render($nazwa, $dane): extract($dane)`, bufor `ob_start()` dla widoku, a wynik (`$tresc`) wstawiony do `views/layout.php`. Układ zawiera menu, komunikat flash i `<link>` do `url('/style.css')`.
3. `FilmController` ma 7 akcji z tabeli tras. Po udanym zapisie — przekierowanie do `/filmy/{id}` z komunikatem flash; przy błędach walidacji formularz z błędami i wpisanymi danymi; nieistniejący film — 404.
4. Widoki: lista (tytuł jako odnośnik do szczegółów, Edytuj, przycisk Usuń w formularzu POST), szczegóły, jeden wspólny formularz dla dodawania i edycji (atrybut `action` z kontrolera), strona 404. Wszystkie dane przez `e()`.

4 Testy i dokumentacja

Aplikację otwieraj pod `http://localhost/katalog_mvc/public/`.

1. Przetestuj wszystkie trasy, adres nieistniejący (`/aktorzy`) i nieistniejący film (`/filmy/999`). W F12 → Network zanotuj kody odpowiedzi (200, 302, 404) dla 5 różnych żądań.
2. Porównaj adresy starej i nowej aplikacji w tabeli (np. `edytuj.php?id=3` → `/filmy/3/edytuj`). Zrób zrzuty listy, formularza z błędami i strony 404, spakuj projekt bez `vendor/` jako `nazwisko_imie_zad17.zip`.

W sprawozdaniu umieść

- ☐ tabela przypisania fragmentów starej aplikacji do warstw M, V, C
- ☐ fragmenty kodu: `.htaccess`, `Router::dispatch()`, jedna akcja kontrolera, `View::render()`
- ☐ tabela adresów: stara aplikacja → nowa aplikacja, z metodą HTTP i kodem odpowiedzi
- ☐ zrzuty: lista, formularz z błędami, 404; archiwum `nazwisko_imie_zad17.zip`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Jakie zadanie ma front controller i co zyskujemy, gdy wszystkie żądania przechodzą przez jeden plik?
2. Dlaczego model nie powinien zawierać HTML, a widok zapytań SQL?
3. Po co w regule `mod_rewrite` warunki `!-f` i `!-d`? Co by się stało z plikiem `style.css` bez nich?
4. Dlaczego usuwanie jest trasą POST, a nie odnośnikiem GET?

Kryteria oceny

Kryterium	Punkty
Struktura katalogów, autoloading, konfiguracja poza <code>public/</code> , tabela M/V/C	3
<code>.htaccess</code> i front controller z 7 trasami	3
Router: parametr <code>{id}</code> , ścieżka bazowa, metoda HTTP, 404	4
Model z zapytaniami przygotowanymi i walidacją; View z układem i flash	4
Kontroler: 7 akcji, PRG, błędy walidacji, 404 dla brakującego filmu	3
Testy z kodami odpowiedzi, zrzuty, archiwum, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

REST API w PHP i klient w JavaScript

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie w PHP interfejsu REST zwracającego JSON dla zasobu z bazy oraz klienta w JavaScript, który korzysta z niego przez `fetch`.

Po zajęciach potrafisz:

- projektuje adresy i metody HTTP dla zasobu (lista, element, dodanie, zmiana, usunięcie)
- zwraca odpowiednie kody stanu (200, 201, 204, 400, 404, 405, 422) i JSON z poprawnymi typami
- odczytuje treść żądania JSON (`php://input`) i waliduje dane
- testuje API poleceniem `curl` lub w kliencie HTTP
- pisze klienta `fetch` z obsługą stanu ładowania i błędów walidacji

Sprzęt i materiały

- komputer z Windows 11, XAMPP, przeglądarka z narzędziami F12
- `r2_17_katalog.sql` i folder `r2_18_start` z Pliki_do_zadan

Oprogramowanie

- **XAMPP**
[apachefriends.org/download.html](https://www.apachefriends.org/download.html)
- **curl** — wbudowany w Windows 11 jako `curl.exe` (w PowerShell pisz `curl.exe`, nie `curl`)
curl.se/windows
- **Postman lub rozszerzenie Thunder Client do VS Code (opcjonalnie)** — Thunder Client: <https://www.thunderclient.com/postman.com/downloads>
- **Visual Studio Code**
code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie z wykładu 2.13 metody HTTP i kody odpowiedzi oraz `fetch` z wykładu 2.03.

Bezpieczeństwo

- API zmieniające dane bez logowania jest dopuszczalne tylko na `localhost` w ćwiczeniu — w prawdziwej aplikacji wymaga uwierzytelnienia (zadanie 32).
- Nie zwracaj w odpowiedziach komunikatów błędów bazy — tylko ogólny komunikat i kod 500; szczegóły do logu.
- Dane z API wstawiaj do strony przez `textContent`, nie `innerHTML`.

Przebieg zadania

1 Projekt API i odczyt danych

Skopiuj `r2_18_start` do `htdocs\katalog_api` i upewnij się, że baza `katalog` istnieje (zadanie 17).

Docelowe zasoby:

GET	<code>api/filmy</code>	lista (opcjonalnie <code>?gatunek=komedia</code>)	200
GET	<code>api/filmy/{id}</code>	jeden film	200 / 404
POST	<code>api/filmy</code>	dodanie (JSON)	201 + Location / 400 / 422
PUT	<code>api/filmy/{id}</code>	podmiana całego filmu	200 / 404 / 422
PATCH	<code>api/filmy/{id}</code>	zmiana wybranych pól	200 / 404 / 422
DELETE	<code>api/filmy/{id}</code>	usunięcie	204 / 404
	inna metoda		405 + nagłówek Allow

1. Utwórz `api/.htaccess` kierujący żądania do `api/index.php` (jak w zadaniu 17). W `api/index.php` ustal ścieżkę zasobu (np. `/filmy/3`) i metodę; dopisz funkcje `odpowiedz(int $kod, mixed $dane)` (kod + `json_encode` z `JSON_UNESCAPED_UNICODE`) i `blad(int $kod, string $komunikat)`.
2. Zaimplementuj `GET /filmy` (z filtrem `?gatunek=` przez zapytanie przygotowane) i `GET /filmy/{id}` (404 z JSON `{"blad": ...}`, gdy brak). Funkcją `film_json()` zamień typy: `id` i `rok` na liczby całkowite, `ocena` na liczbę lub `null`.
3. Otwórz `http://localhost/katalog_api/api/filmy` w przeglądarce, a potem w konsoli: `curl.exe -i http://localhost/katalog_api/api/filmy/3`. Zanotuj nagłówki `Content-Type` i kod odpowiedzi.

2 Zapis danych: POST, PUT, PATCH, DELETE

Treść JSON odczytasz przez `json_decode(file_get_contents('php://input'), true)`; jeśli wynik nie jest tablicą → 400.

```
curl.exe -i -X POST -H "Content-Type: application/json" -d @film.json
http://localhost/katalog_api/api/filmy
curl.exe -i -X PATCH -H "Content-Type: application/json" -d @zmiana.json
http://localhost/katalog_api/api/filmy/9
curl.exe -i -X DELETE http://localhost/katalog_api/api/filmy/9
```

1. Napisz `waliduj(array $f): array` zwracającą błędy w postaci `pole => komunikat` (tytuł 1–120, reżyser 1–80, rok — liczba całkowita 1895–2030, gatunek z listy, ocena 1–10 lub `null`). Błędy → 422 z `{"blad": ..., "bledy": {...}}`.
2. POST: zapis, kod 201, nagłówek `Location` z adresem nowego filmu i JSON utworzonego rekordu. PUT: wymaga wszystkich pól. PATCH: połącz dane z bazy z przesłanymi (`array_merge`) i zwaliduj całość. DELETE: 204 bez treści.
3. Dla innych metod zwróć 405 z nagłówkiem `Allow`. Błędy PDO przechwyc (`try / catch`) i zwróć 500 z ogólnym komunikatem.
4. Przetestuj curl (ramka) wszystkie przypadki z tabeli zasobów, także błędne: zły JSON, puste pola, nieistniejące id, `DELETE` na `/filmy`. Wyniki wpisz do tabeli w sprawozdaniu.

3 Klient w JavaScript

Plik `index.html` jest gotowy (tabela `#lista`, pole `#stan`, filtr `#filtr`, formularz `#formularz`, lista `#bledy`). Kod pisz w `app.js` (moduł).

1. Funkcja `async wczytaj()`: pokaż „Ładowanie...”, pobierz listę (z filtrem, `encodeURIComponent`), sprawdź `odp.ok`, zbuduj wiersze tabeli przez `createElement` i `textContent`, pokaż liczbę filmów; błąd sieci lub HTTP — komunikat w `#stan`.
2. Obsłuż formularz: `preventDefault()`, zbuduj obiekt z `FormData` (rok jako liczba, pusta ocena jako `null`), wyślij `POST` z nagłówkiem `Content-Type: application/json`. Kod 201 → wyczyść formularz i odśwież listę; 422 → wypisz błędy pól w `#bledy`.
3. Dodaj przycisk „Usuń” w każdym wierszu: `confirm()`, żądanie `DELETE`, przy 204 odśwież listę. Zmiana filtra odświeża listę.
4. W F12 → Network obejrzyj żądania klienta (metoda, kod, nagłówki, Payload, Response). Zrób zrzuty: aplikacja, błędy walidacji, karta Network z żądaniem POST. Spakuj projekt jako `nazwisko_imie_zad18.zip` — użyjesz go w zadaniu 23.

W sprawozdaniu umieść

- ☐ tabela testów API: metoda, adres, dane, oczekiwany i otrzymany kod
- ☐ fragmenty kodu: rozpoznanie trasy, POST z walidacją, PATCH, funkcja `wczytaj()`
- ☐ zrzuty: odpowiedź curl z nagłówkami dla 201, aplikacja klienta, błędy 422, karta Network
- ☐ archiwum `nazwisko_imie_zad18.zip` i odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różni się PUT od PATCH? Który z nich wykonany dwa razy daje ten sam wynik?
2. Dlaczego po dodaniu zasobu zwracamy 201 i nagłówek `Location`, a po usunięciu 204?
3. Kiedy zwrócić 400, a kiedy 422?
4. Dlaczego `fetch` nie zgłasza wyjątku przy odpowiedzi 404 i jak to obsłużyć?
5. Po co w API nagłówki CORS i dlaczego klient z tego samego adresu ich nie potrzebuje?

Kryteria oceny

Kryterium	Punkty
GET listy (z filtrem) i elementu, typy JSON, 404 z komunikatem	4
POST z walidacją: 201 + Location, 400, 422 z błędami pól	4
PUT, PATCH, DELETE (204), 405 + Allow, obsługa błędów 500	4
Tabela testów curl z kodami	2
Klient: lista z ładowaniem i błędem, filtr, dodawanie z błędami 422, usuwanie	4
Zrzuty, archiwum, odpowiedzi na pytania	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Próba INF.03: pełny arkusz — Pasieka Żłoty UI

Czas: 2 × 45 min

praca samodzielna (warunki egzaminacyjne)

Cel: Sprawdzenie w warunkach zbliżonych do egzaminu umiejętności wykonania pełnego zadania INF.03: baza danych, witryna HTML/CSS, skrypt PHP z MySQLi i skrypt JavaScript.

Po zajęciach potrafisz:

- importuje bazę i zapisuje 4 zapytania SQL zgodnie z poleceniami
- buduje witrynę o zadanym układzie bloków i formatowaniu
- generuje zawartość strony z bazy skryptem PHP (mysqli) i zamyka połączenie
- pisze funkcję JavaScript obliczającą wynik na podstawie formularza
- pracuje samodzielnie w limicie czasu i sprawdza pracę według kryteriów

Sprzęt i materiały

- komputer z Windows 11, XAMPP, edytor kodu bez rozszerzeń generujących kod (np. VS Code bez AI, Notepad++)
- folder `r2_19_pasieka` z Pliki_do_zadan

Oprogramowanie

- **XAMPP** — Apache, MariaDB, phpMyAdmin
[apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code lub Notepad++** —
Notepad++: <https://notepad-plus-plus.org/code.visualstudio.com>

Przygotowanie przed zajęciami

- Powtórz zapytania z JOIN i GROUP BY, funkcje mysqli z zadania 8 oraz układ bloków float/flex.

Bezpieczeństwo

- Pracuj samodzielnie, bez internetu i narzędzi AI — tak jak na egzaminie. Wyniki zapisuj tylko w folderze `nazwisko_imie` na pulpicie.

Przebieg zadania

1 Baza danych (ok. 20 min)

Utwórz na pulpicie folder `nazwisko_imie`, skopiuj do niego grafiki. W phpMyAdmin zaimportuj `pasieka.sql` (tworzy bazę `pasieka`: tabele `miody` i `zamowienia`). Zapytania zapisz w pliku `kwerendy.txt`, a zrzut wyniku każdego — jako `kw1.png` ... `kw4.png`.

1. Zapytanie 1: nazwa, waga i cena miodów, których stan jest większy od 0, posortowane rosnąco według nazwy, a przy tej samej nazwie według wagi.
2. Zapytanie 2: nazwa i cena miodów w słoikach o wadze 1000 g.
3. Zapytanie 3: nazwa i waga miodu oraz łączna liczba sprzedanych słoików (alias `sprzedano`) — zapytanie do tabel w relacji, grupowanie według miodu, sortowanie malejąco według `sprzedano`, a przy równej liczbie alfabetycznie według nazwy.
4. Zapytanie 4: dodaj do tabeli `zamowienia` kolumnę `telefon` typu tekstowego o długości 15 znaków.

2 Witryna (ok. 40 min)

Plik `pasieka.php`, tytuł karty „Pasieka Żłoty Ul”, język polski, arkusz `styl.css`. Układ: baner na całej szerokości, pod nim trzy bloki obok siebie (lewy 25%, środkowy 50%, prawy 25%, każdy wysokości 650 px), na dole stopka.

1. Baner: obraz `sloik.png` z tekstem alternatywnym „Słoik miodu” (po lewej, wysokość 100 px) i nagłówek h1 „Pasieka Żłoty Ul”. Blok lewy: h2 „Nasze miody” i lista numerowana wypełniana skryptem 1.
2. Blok środkowy: obraz `ul.png` (alt „Ul w pasiece”), h2 „Najczęściej kupowane”, tabela z nagłówkami Miód, Waga, Sprzedano (słoiki) wypełniana skryptem 2.
3. Blok prawy: h2 „Kalkulator”, etykieta „Liczba słoików:” i pole liczbowe `sloiki` (wartość 1), dwa przyciski opcji `waga`: „500 g – 34 zł” (zaznaczony) i „1000 g – 62 zł” (wartości 34 i 62), przycisk „Oblicz”, pusty akapit `wynik`. Stopka: akapit „Stronę wykonał: ” i Twoje imię i nazwisko.
4. Style: krój Verdana dla wszystkich elementów; tło strony #FFF8E1; baner tło #F2A900, kolor tekstu #3E2723, wyrównanie do środka, wysokość 120 px; h1 rozmiar 260%; blok lewy tło #FFE082, prawy #FFECB3; obraz w bloku środkowym szerokość 100%, zaokrąglenie narożników 10 px; tabela na całą szerokość bloku, krawędzie komórek 1px solid #8D6E63, wiersz pod kursorem tło #F2A900; przycisk tło #3E2723, tekst biały, bez krawędzi; stopka tło #3E2723, tekst biały, wyrównanie do prawej.

3 Skrypty PHP i JavaScript (ok. 25 min)

1. Skrypt 1 (blok lewy): połącz się z bazą `pasieka` (`mysqli_connect`), wykonaj zapytanie 1 i dla każdego wiersza wypisz element listy w formacie „Miód lipowy 500 g – 34.00 zł”.
2. Skrypt 2 (blok środkowy): wykonaj zapytanie 3 i dla każdego wiersza wypisz wiersz tabeli z trzema komórkami (waga z jednostką „g”). Na końcu zamknij połączenie z bazą.
3. JavaScript (plik `skrypt.js`): funkcja wywoływana przyciskiem „Oblicz” odczytuje liczbę słoików i cenę zaznaczonej opcji, oblicza koszt; od 5 słoików udziela rabatu 10%. W akapicie `wynik` wypisuje „Do zapłaty: X zł” (2 miejsca po przecinku), z rabatem z dopiskiem „(rabat 10%)”; dla liczby mniejszej niż 1 — „Podaj liczbę słoików (co najmniej 1)”.
4. Zapisz zrzut całej strony jako `strona.png`. Sprawdź pracę według kryteriów oceny i wpisz samoocenę do sprawozdania.

W sprawozdaniu umieść

- ☐ folder `nazwisko_imie`: `kwerendy.txt`, `kw1.png` – `kw4.png`, `pasieka.php`, `styl.css`, `skrypt.js`, grafiki, `strona.png`
- ☐ tabela samooceny według kryteriów (punkty przyznane sobie i nauczyciela)
- ☐ lista błędów i trudności oraz plan, co powtórzyć przed egzaminem

Pytania kontrolne

1. Która część arkusza zajęła Ci najwięcej czasu i dlaczego?
2. Czym różni się `mysqli_fetch_row` od `mysqli_fetch_assoc`? Której funkcji użyłeś i gdzie?
3. Dlaczego w zapytaniu 3 grupujemy według `miody.id`, a nie tylko według nazwy?

Kryteria oceny

Kryterium	Punkty
Zapytania 1–4 poprawne, zrzuty wyników	4
Struktura HTML: baner, 3 bloki, stopka, obrazy z alt, lista, tabela, formularz kalkulatora	4
Style CSS zgodne z poleceniem (kolory, wymiary, układ, hover, krój)	4
Skrypt 1 i skrypt 2 (mysqli, pętla, format, zamknięcie połączenia)	4
JavaScript: odczyt pól, rabat, format wyniku, komunikat błędu	3
Samooocena i analiza błędów	1
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Laravel — pierwsza aplikacja

Czas: 2 × 45 min

praca indywidualna

Cel: Utworzenie pierwszej aplikacji w frameworku Laravel: trasy, kontroler, szablony Blade z układem, migracja, model Eloquent i lista rekordów z bazy.

Po zajęciach potrafisz:

- tworzy projekt Laravel i uruchamia serwer deweloperski
- definiuje trasy z nazwami i parametrami, w tym route model binding
- buduje widoki Blade dziedziczące wspólny układ (`@extends`, `@section`, `@yield`)
- tworzy tabelę migracją, wypełnia ją seederem i pobiera dane modelem Eloquent
- posługuje się poleceniami `php artisan` i rozpoznaje strukturę katalogów projektu

Sprzęt i materiały

- komputer z Windows 11, dostęp do internetu (pobranie pakietów Composera)
- PHP ≥ 8.3 z Composerem (Laravel Herd lub php.new) albo XAMPP (PHP 8.2 → Laravel 12)

Oprogramowanie

- **Laravel Herd dla Windows (zalecane, bezpłatny)** — zawiera PHP, Composer i instalator Laravel
herd.laravel.com/windows
- **Dokumentacja Laravel — instalacja** — alternatywa: skrypt `php.new` (PowerShell jako administrator)
laravel.com/docs/13.x/installation
- **Composer** — przy pracy na PHP z XAMPP
getcomposer.org/download
- **Visual Studio Code** — rozszerzenie Laravel (oficjalne) podświetla Blade
code.visualstudio.com

Przygotowanie przed zajęciami

- Nauczyciel instaluje wcześniej Herd lub PHP 8.3+ z Composerem. Sprawdź: `php -v`, `composer -V`, `laravel --version`.

Bezpieczeństwo

- Plik `.env` zawiera klucz aplikacji `APP_KEY` i dane bazy — nie udostępniaj go i nie wysyłaj do repozytorium (jest w `.gitignore`).
- Tryb `APP_DEBUG=true` jest tylko do pracy lokalnej; na serwerze produkcyjnym musi być `false`.

Przebieg zadania

1 Nowy projekt

Z Herd: `cd ~\Herd`, potem `laravel new filmoteka` (instalator pyta m.in. o zestaw startowy — wybierz **None**, o bazę — wybierz **SQLite**; pytania mogą się różnić między wersjami). Bez instalatora: `composer create-project laravel/laravel filmoteka` — przy PHP 8.2 z XAMPP Composer zainstaluje Laravel 12; kod zadania działa w obu wersjach.

1. Utwórz projekt i uruchom go: Herd — `http://filmoteka.test`; w innym przypadku `php artisan serve` i `http://127.0.0.1:8000`. Zanotuj wersję: `php artisan --version`.
2. Opisz w sprawozdaniu rolę katalogów i plików: `app/Models`, `app/Http/Controllers`, `routes/web.php`, `resources/views`, `database/migrations`, `public`, `.env`, `vendor`. Obejrzyj listę tras: `php artisan route:list`.

2 Trasy i widoki Blade

Szablony zapisuj w `resources/views` z rozszerzeniem `.blade.php`. `{{ $x }}` wypisuje wartość z automatycznym escapowaniem.

1. Utwórz układ `resources/views/layouts/app.blade.php`: nagłówek „Filmoteka”, menu (Start, Filmy, O nas — adresy przez `route('nazwa')`, aktywna pozycja wyróżniona dyrektywą `@class` i `request()->routeIs(...)`, `@yield('content')`, stopka z wersją Laravel (`app()->version()`). Styl: `public/css/style.css` dołączony przez `asset('css/style.css')`.
2. W `routes/web.php` zamień trasę `/` tak, by zwracała widok `start` z przekazaną zmienną `dzisiaj` (bieżąca data `now()->format('d.m.Y')`); nadaj trasie nazwę `start`. Dodaj trasę `/o-nas` (nazwa `o-nas`) przekazującą tablicę 3 autorów, wypisaną w widoku pętlą `@foreach`.
3. Oba widoki dziedziczą układ (`@extends('layouts.app')`, `@section('title', ...)`, `@section('content')` ... `@endsection`). Sprawdź, co pokaże strona, gdy jeden z autorów będzie miał w nazwie znaki `<b>`.

3 Migracja, model i dane

1. Wykonaj `php artisan make:model Film -m`. Zmień nazwę tabeli w migracji na `filmy` i dodaj kolumny: `tytuł` (string 120), `rezyser` (string 80), `rok` (unsignedSmallInteger), `gatunek` (string 20), `ocena` (decimal 3,1, nullable) oraz `timestamps()`. W modelu ustaw `protected $table = 'filmy';`, `$fillable` i stałą `GATUNKI`.
2. Uruchom `php artisan migrate` i obejrzyj tabelę (SQLite: rozszerzenie bazy w VS Code lub `php artisan db:table filmy`). Dlaczego bez `$table` Laravel szukałby tabeli `films`?
3. Utwórz seeder `php artisan make:seeder FilmSeeder` dodający 8 filmów przez `Film::create([...])` (np. z zadania 17), wywołaj go w `DatabaseSeeder` i wykonaj `php artisan db:seed`. Sprawdź liczbę rekordów w `php artisan tinker: App\Models\Film::count()`.

4 Kontroler i lista z bazy

1. Wykonaj `php artisan make:controller FilmController`. Metoda `index(Request $request)` pobiera filmy posortowane po tytule, opcjonalnie filtrowane gatunkiem z `?gatunek=` (`when(...)`), i przekazuje je do widoku `filmy.index` razem z listą gatunków. Metoda `show(Film $film)` zwraca widok `filmy.show`.
2. Dodaj trasy `GET /filmy` (nazwa `filmy.index`) i `GET /filmy/{film}` (nazwa `filmy.show`). Sprawdź adres nieistniejącego filmu, np. `/filmy/999` — jaki kod zwraca Laravel i dlaczego?
3. Widok `filmy/index.blade.php`: formularz GET z listą gatunków (`@selected`), liczba filmów, tabela z tytułem jako odnośnikiem `route('filmy.show', $film)`, `@if ($filmy->isEmpty())` z komunikatem. Widok `filmy/show.blade.php`: szczegóły i data dodania `created_at->format('d.m.Y H:i')`.
4. Zrób zrzuty: start, o nas, lista, lista z filtrem, szczegóły, wynik `route:list`. Spakuj projekt **bez** `vendor/` i `node_modules/` jako `nazwisko_imie_zad20.zip` — będzie potrzebny w zadaniu 21.

W sprawozdaniu umieść

- ☐ tabela wykonanych poleceń (krok.punkt) z poleceniami artisan i fragmentami kodu
- ☐ opis katalogów projektu i wersja Laravel/PHP
- ☐ zrzuty: strony aplikacji, `route:list`, wynik `Film::count()`
- ☐ archiwum `nazwisko_imie_zad20.zip` i odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Co robi migracja i dlaczego jest lepsza od ręcznego tworzenia tabeli w phpMyAdmin przy pracy zespołowej?
2. Czym jest route model binding i ile linii kodu z zadania 17 zastępuje?
3. Dlaczego `{{ $autor }}` jest bezpieczne, a `{!! $autor !!}` wymaga ostrożności?
4. Do czego służą nazwy tras (`->name(...)`) i funkcja `route()` ?

Kryteria oceny

Kryterium	Punkty
Projekt uruchomiony, opis struktury katalogów, <code>route:list</code>	3
Układ Blade z menu (aktywna pozycja) i 2 widoki dziedziczące z przekazanymi danymi	4
Migracja tabeli <code>filmy</code> , model z <code>\$table</code> i <code>\$fillable</code> , seeder z 8 rekordami	5
Kontroler: lista z filtrem gatunku i szczegóły (route model binding, 404)	5
Zrzuty, archiwum, odpowiedzi na pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Laravel — formularze, walidacja i CRUD

Czas: 2 × 45 min

praca indywidualna

Cel: Rozbudowa aplikacji Laravel o pełny CRUD z kontrolerem zasobów, walidacją danych, komunikatami błędów i paginacją oraz porównanie z czystym PHP.

Po zajęciach potrafisz:

- tworzy kontroler zasobów i rejestruje 7 tras jednym wywołaniem `Route::resource`
- buduje formularze z `@csrf` i `@method('PUT'/'DELETE')`
- waliduje dane `$request->validate()` i wyświetla błędy przy polach (`@error`, `old()`)
- zapisuje, zmienia i usuwa rekordy modelem Eloquent z komunikatem flash
- dzieli listę na strony (`paginate`) i porównuje nakład pracy z aplikacją w czystym PHP

Sprzęt i materiały

- komputer z Windows 11 i środowiskiem z zadania 20 (Herd lub PHP + Composer)
- projekt `filmoteka` z zadania 20

Oprogramowanie

- **Laravel Herd dla Windows lub PHP + Composer** — jak w zadaniu 20
herd.laravel.com/windows
- **Dokumentacja Laravel — walidacja** — lista reguł: Available Validation Rules
laravel.com/docs/13.x/validation
- **Visual Studio Code**
code.visualstudio.com

Przygotowanie przed zajęciami

- Uruchom projekt z zadania 20 i sprawdź, że lista filmów działa.

Bezpieczeństwo

- Nie wyłączaj ochrony CSRF — każdy formularz POST/PUT/DELETE musi zawierać `@csrf` (bez tokenu Laravel zwraca błąd 419).
- Do `create()` i `update()` przekazuj tylko dane zwrócone przez `validate()` — model z `$fillable` przyjmie wyłącznie dozwolone pola.

Przebieg zadania

1 Kontroler zasobów i trasy

W `routes/web.php` zastąp dwie trasy filmów jedną:

```
Route::resource('filmy', FilmController::class)->parameters(['filmy' => 'film']);
```

1. Uruchom `php artisan route:list --name=filmy` i przepisz do sprawozdania 7 tras (metoda, adres, nazwa, akcja). Po co `parameters(['filmy' => 'film'])`?
2. Dopisz w kontrolerze brakujące metody: `create`, `store`, `edit`, `update`, `destroy` (możesz porównać z szablonem z `php artisan make:controller TestController --resource --model=Film` i usunąć potem plik testowy).

2 Dodawanie z walidacją

Reguły zapisz w prywatnej metodzie `reguly()`, aby użyć ich w `store` i `update`.

```
'tytul' => ['required', 'string', 'max:120'],
'rezyser' => ['required', 'string', 'max:80'],
'rok' => ['required', 'integer', 'between:1895,2030'],
'gatunek' => ['required', Rule::in(Film::GATUNKI)],
'ocena' => ['nullable', 'numeric', 'between:1,10'],
```

1. Utwórz częściowy widok `filmy/_form.blade.php` z polami formularza: `@csrf`, wartości `old('pole', $film->pole)`, pod każdym polem `@error('pole') <span class="blad">{{ $message }}</span> @enderror`, lista gatunków z `@selected`.
2. Widok `filmy/create.blade.php` dołącza formularz (`@include`) z akcją `route('filmy.store')`; nad formularzem pokaż liczbę błędów (`$errors->count()`). Metoda `create()` przekazuje pusty obiekt `new Film()` i listę gatunków.
3. `store()`: `$dane = $request->validate($this->reguly());`, `Film::create($dane)`, przekierowanie do szczegółów z `->with('status', 'Dodano film.')`. W widokach `index` i `show` wyświetl `session('status')`.
4. Wyślij pusty formularz i formularz z rokiem 3000 — zanotuj komunikaty. Ustaw w `.env` `APP_LOCALE=pl` i utwórz `lang/pl/validation.php` z polskimi komunikatami dla użytych reguł i nazwami pól w `attributes` (pełne tłumaczenia: pakiet `laravel-lang/lang`).

3 Edycja, usuwanie i paginacja

1. Widok `filmy/edit.blade.php` z tym samym `_form`, akcją `route('filmy.update', $film)` i `@method('PUT')`. `update()` waliduje i wywołuje `$film->update(...)`.
2. W liście dodaj przy każdym filmie „Edytuj” i formularz usuwania (`@csrf`, `@method('DELETE')`, potwierdzenie `confirm`). `destroy()` usuwa rekord i wraca do listy z komunikatem zawierającym tytuł.
3. Zamień `get()` w `index()` na `paginate(5)->withQueryString()`, pod tabelą wypisz `{{ $filmy->links('pagination::default') }}` (style klasy `.pagination` w CSS), a liczbę filmów pobierz `$filmy->total()`. Sprawdź filtr gatunku na stronie 2.
4. Test 419: w F12 → Elements usuń z formularza ukryte pole `_token` i wyślij formularz. Zanotuj wynik.

4 Porównanie z czystym PHP

1. Policz niepuste linie kodu odpowiadających sobie części w zadaniu 17 (lub 9) i w Laravel (kontroler, model, trasy, widoki filmów, układ). Uzupełnij tabelę porównania w sprawozdaniu: walidacja, ochrona CSRF, 404 dla brakującego rekordu, komunikaty flash, paginacja, escapowanie — jak zrealizowane w czystym PHP, a jak w Laravel.
2. Zrób zrzuty: lista z paginacją i komunikatem, formularz z błędami (po polsku), edycja, strona 419. Spakuj projekt bez `vendor/` i `node_modules/` jako `nazwisko_imie_zad21.zip`.

W sprawozdaniu umieść

- ☐ tabela 7 tras resource (metoda, adres, nazwa, akcja)
- ☐ fragmenty kodu: reguły walidacji, `store`, `update`, formularz usuwania
- ☐ tabela porównania czysty PHP / Laravel z liczbą linii
- ☐ zrzuty: lista z paginacją, błędy walidacji, edycja, 419; archiwum `nazwisko_imie_zad21.zip`
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Dlaczego formularz HTML wysyła PUT i DELETE jako POST z polem `_method`?

2. Co dzieje się po nieudanej walidacji w `$request->validate()` (przekierowanie, sesja, `old()`, `$errors`)?
3. Przed czym chroni `$fillable` i dlaczego do `create()` przekazujemy wynik `validate()`, a nie `$request->all()`?
4. Jakie są zalety i wady frameworka w porównaniu z własnym kodem z zadania 17?

Kryteria oceny

Kryterium	Punkty
<code>Route::resource</code> z parametrem, tabela tras	2
Formularz <code>_form</code> z <code>@csrf</code> , <code>old()</code> , <code>@error</code> ; dodawanie z walidacją 5 pól i flash	5
Polskie komunikaty walidacji (<code>APP_LOCALE</code> , <code>lang/pl</code>)	2
Edycja (PUT) i usuwanie (DELETE) z potwierdzeniem i komunikatem	4
Paginacja z zachowaniem filtra, test 419	3
Porównanie z czystym PHP, zrzuty, archiwum, pytania	4
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

React — komponenty i stan

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie pierwszej aplikacji w React: podział interfejsu na komponenty, przekazywanie danych przez props, stan `useState`, renderowanie list z kluczem i formularz kontrolowany.

Po zajęciach potrafisz:

- tworzy projekt React w Vite i uruchamia serwer deweloperski
- dzieli interfejs na komponenty funkcyjne i przekazuje im dane oraz funkcje przez props
- przechowuje dane w stanie `useState` i zmienia go niemutowalnie (nowa tablica / nowy obiekt)
- renderuje listy przez `map()` z unikalnym `key` i stosuje renderowanie warunkowe
- obsługuje formularz kontrolowany z walidacją

Sprzęt i materiały

- komputer z Windows 11, Node.js LTS, dostęp do internetu (npm)
- plik `r2_22_index.css` z Pliki_do_zadan

Oprogramowanie

- **Node.js LTS** — Vite 8 wymaga Node.js 20.19+ lub 22.12+; sprawdź `node -v`
nodejs.org
- **Vite — dokumentacja**
vite.dev/guide
- **React — dokumentacja** — rozdziały Describing the UI i Adding Interactivity
react.dev/learn
- **Visual Studio Code + React Developer Tools** — rozszerzenie przeglądarki do podglądu komponentów i stanu
react.dev/learn/react-developer-tools

Przygotowanie przed zajęciami

- Przejrzyj wykład 2.15 (JSX, props, `useState`) i powtórz z wykładu 2.01 funkcje strzałkowe, destrukuryzację, spread, `map` i `filter`.

Bezpieczeństwo

- Instaluj pakiety tylko poleceniami z instrukcji; po instalacji sprawdź komunikat `npm audit` wyświetlany przez npm.
- Folder `node_modules` nie trafia do archiwum ani repozytorium (jest w `.gitignore`).

Przebieg zadania

1 Projekt Vite + React

```
npm create vite@latest planer -- --template react
cd planer
npm install
npm run dev
```

1. Utwórz projekt poleceniami z ramki (jeśli pojawią się pytania, wybierz React i wariant JavaScript). Otwórz adres podany przez Vite (domyślnie `http://localhost:5173`). Zanotuj wersje React i Vite z `package.json`.
2. Opisz rolę plików: `index.html`, `src/main.jsx`, `src/App.jsx`, `package.json`, `vite.config.js`. Zmień tekst w `App.jsx` i zaobserwuj odświeżenie bez przeładowania strony (HMR).
3. Wyczyść szablon: usuń zawartość `App.jsx` i plik `App.css`, zastąp `src/index.css` plikiem `r2_22_index.css`, ustaw w `index.html` `lang="pl"` i tytuł „Planer ucznia”. Komponenty twórz w folderze `src/components`.

2 Komponenty i props

Każdy komponent to funkcja z wielkiej litery w osobnym pliku `.jsx` z `export default`.

1. Komponent `Naglowek` przyjmuje props `tytuł` i `podtytuł` (wartość domyślna „React + Vite” w destrukuryzacji) i renderuje `<header className="naglowek">` z `h1` i akapitem. Użyj go w `App`.
2. Komponent `Licznik` przyjmuje `etykieta`, `start` (0), `krok` (1), `max` (10) i ma stan `wartosc`. Przyciski: `-krok`, `+krok` (wartość nie spada poniżej 0 ani nie przekracza `max`; przycisk wyłączony `disabled` na granicy), „Zeruj”. Po osiągnięciu maksimum wyświetl komunikat (renderowanie warunkowe `&&`).
3. Umieść w `App` dwa liczniki z różnymi props: „Szkłanki wody” (`max` 8) i „Minuty nauki” (`krok` 15, `max` 120). Sprawdź, że każdy ma własny stan. W React Developer Tools obejrzyj props i stan liczników.

3 Lista zadań

Stan listy trzymaj w komponencie `ListaZadan`; zadanie to obiekt `{ id, tresc, priorytet, zrobione }`. Zaczynasz od 3 przykładowych zadań.

```
setZadania((z) => [...z, nowe]); // dodanie
setZadania((z) => z.map((x) => x.id === id ? { ...x, zrobione: !x.zrobione } : x));
setZadania((z) => z.filter((x) => x.id !== id)); // usunięcie
```

1. Komponent `Zadanie` (props: `zadanie`, `onPrzelacz`, `onUsun`) renderuje `<li>` z polem wyboru, treścią, priorytetem i przyciskiem usuwania. Klasy CSS: priorytet (`niski` / `normalny` / `wysoki`) oraz `zrobione`.
2. W `ListaZadan` wyrenderuj zadania `map()` z `key={z.id}`. Napisz funkcje `przelacz` i `usun` według ramki i przekaż je do `Zadanie`. Pod listą: „Do zrobienia: X z Y”.
3. Dodaj trzy przyciski filtra (wszystkie, do zrobienia, zrobione) — stan `filtr`, aktywny przycisk z klasą `aktywny`. Gdy lista po filtrze jest pusta, wyświetl „Brak zadań w tym widoku”. Sprawdź w konsoli F12, co się stanie, gdy na chwilę usuniesz `key` (zanotuj ostrzeżenie).

4 Formularz kontrolowany

1. Komponent `FormularzZadania` (prop `onDodaj`) ma stan `tresc`, `priorytet`, `bład`. Pola mają `value` i `onChange` (formularz kontrolowany); pod polem licznik znaków „N/80”, przycisk „Dodaj” wyłączony, gdy pole jest puste.
2. Po wysłaniu (`onSubmit`, `preventDefault`): treść krótsza niż 3 znaki → komunikat błędu; poprawna → `onDodaj(tresc, priorytet)`, wyczyszczenie pól i błędu. Nowe zadanie dostaje `id` z `crypto.randomUUID()`.
3. Wykonaj `npm run build` i obejrzyj folder `dist`. Zrób zrzuty: aplikacja z licznikiem na maksimum, błąd formularza, lista po filtrze, React Developer Tools. Spakuj projekt **bez** `node_modules` i `dist` jako `nazwisko_imie_zad22.zip` — użyjesz go w zadaniu 23.

W sprawozdaniu umieść

- ☐ tabela wykonanych poleceń (krok.punkt) z fragmentami kodu
- ☐ opis plików projektu Vite i wersje React/Vite
- ☐ drzewo komponentów aplikacji z zaznaczeniem, gdzie jest stan, a co przechodzi przez props
- ☐ zrzuty: aplikacja, błąd formularza, filtr, React Developer Tools, ostrzeżenie o braku `key`
- ☐ archiwum `nazwisko_imie_zad22.zip` i odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Czym różnią się props od stanu? Kto może je zmieniać?
2. Dlaczego nie wolno zmieniać stanu przez `zadania.push(...)` i po co forma `setZadania((z) => ...)`?
3. Do czego React używa atrybutu `key` i dlaczego indeks tablicy jest złym kluczem dla listy z usuwaniem?
4. Co oznacza „formularz kontrolowany” i jakie daje korzyści w porównaniu z odczytem pól w zdarzeniu submit?
5. Porównaj dodawanie elementu listy w React i w czystym JS z DOM (zadanie 3).

Kryteria oceny

Kryterium	Punkty
Projekt Vite uruchomiony, opis plików, wyczyszczony szablon	2
Komponenty <code>Naglowek</code> i <code>Licznik</code> z props i wartościami domyślnymi; dwa niezależne liczniki z limitami	5
Lista zadań: <code>map</code> z <code>key</code> , przełączanie, usuwanie bez mutacji, podsumowanie, filtr	6
Formularz kontrolowany z licznikiem znaków, walidacją i czyszczeniem	4
Build, drzewo komponentów, zrzuty, archiwum, pytania	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

React — dane z API

Czas: 2 × 45 min

praca indywidualna

Cel: Pobieranie danych z API w aplikacji React (useEffect + fetch) z obsługą stanu ładowania i błędu, filtrowanie i sortowanie po stronie klienta oraz przygotowanie wersji produkcyjnej.

Po zajęciach potrafisz:

- pobiera dane z API w hooku useEffect z poprawną tablicą zależności i sprzątniem (AbortController)
- obsługuje w interfejsie trzy stany: ładowanie, błąd i dane
- filtruje i sortuje listę na podstawie stanu formularza kontrolowanego bez ponownego pobierania danych
- dzieli interfejs na komponenty z props
- buduje wersję produkcyjną (`npm run build`) i uruchamia folder dist

Sprzęt i materiały

- komputer z Windows 11, XAMPP (Apache) i Node.js LTS
- folder `r2_23_api` z Pliki_do_zadan (plik `api.php` i dane `gry.json`)
- projekt React z zadania 22 lub nowy projekt Vite

Oprogramowanie

- **Node.js LTS** — zawiera npm; sprawdź `node -v` i `npm -v`
nodejs.org
- **Visual Studio Code** — terminal wbudowany: Terminal → New Terminal
code.visualstudio.com
- **XAMPP** — Apache serwuje API pod `http://localhost/r2_23_api/api.php`
apachefriends.org/download.html
- **React DevTools (opcjonalnie)** — podgląd stanu komponentów w F12
react.dev/learn/react-developer-tools

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 2.15 hooki useState i useEffect oraz z zadania 4 obsługę fetch i async/await.
- Upewnij się, że projekt z zadania 22 uruchamia się poleceniem `npm run dev`.

Bezpieczeństwo

- Nie umieszczaj w kodzie frontendu haseł ani kluczy API — wszystko, co trafia do folderu dist, może przeczytać każdy użytkownik.
- Wyświetlaj dane w JSX zwykłymi wyrażeniami `{...}` (React sam je escapuje); nie używaj dangerouslySetInnerHTML.

Przebieg zadania

1 Przygotowanie API i projektu

API zwraca tablicę obiektów: id, nazwa, kategoria, min_graczy, max_graczy, czas_min, wiek_od, ocena, cena, dostepna. Parametry testowe: `?opoznienie=2` (odpowiedź po 2 s) i `?blad=1` (kod 500).

```
npm create vite@latest katalog-gier -- --template react
cd katalog-gier
npm install
npm run dev
```

1. Skopiuj folder `r2_23_api` do `C:\xampp\htdocs`, uruchom Apache i otwórz `http://localhost/r2_23_api/api.php`. Sprawdź w F12 → Network kod odpowiedzi i nagłówek Content-Type.
2. Utwórz projekt poleceniami z ramki, usuń przykładową zawartość `App.jsx` i `App.css`, ustaw `lang="pl"` i tytuł strony w `index.html`.
3. Utwórz plik `.env` z wierszem `VITE_API_URL=http://localhost/r2_23_api/api.php` i odczytaj adres w kodzie przez `import.meta.env.VITE_API_URL`. Po zmianie `.env` uruchom ponownie `npm run dev`.

2 Pobieranie danych w useEffect

Stan aplikacji: `gry` (tablica), `ladowanie` (true/false), `blad` (null albo tekst).

1. W `App.jsx` utwórz trzy stany `useState` i w `useEffect` z pustą tablicą zależności pobierz dane funkcją async (`fetch + await odp.json()`). Wyświetl na razie same nazwy gier w liście `<ul>` z atrybutem `key={g.id}`.
2. Sprawdź `odp.ok` i przy błędzie HTTP zgłoś wyjątek z kodem odpowiedzi; w bloku catch zapisz komunikat do stanu `blad`, a w finally wyłącz `ladowanie`.
3. Wyświetlaj warunkowo: „Ładowanie danych...” (`role="status"`), komunikat błędu (`role="alert"`) albo listę. Sprawdź oba stany, dopisując do adresu API `?opoznienie=2`, a potem `?blad=1`; zrób zrzuty.
4. Dodaj przycisk „Spróbuj ponownie” przy błędzie: zwiększa licznik w stanie `proba`, który dopisz do tablicy zależności `useEffect`.
5. Dodaj sprzątanie efektu: utwórz `AbortController`, przekaż `signal` do `fetch` i zwróć z efektu funkcję `() => kontroler.abort()`. Wyjaśnij w sprawozdaniu, dlaczego w trybie StrictMode w F12 → Network widać dwa żądania.

3 Komponenty i props

1. Utwórz komponent `GraKarta` (folder `src/components`) wyświetlający kartę gry: kategorię, nazwę, liczbę graczy, czas, wiek, cenę w zł (`Intl.NumberFormat('pl-PL', {style: 'currency', currency: 'PLN'})`) i ocenę.
2. Gry niedostępne (`dostepna: false`) wyróżnij inną klasą CSS i dopiskiem „chwilowo niedostępna”. Wyświetl karty w siatce CSS Grid (auto-fill, min. 230 px).

4 Filtrowanie i sortowanie po stronie klienta

Dane pobierasz raz — filtry tylko przeliczają widoczną listę przy każdym renderowaniu. Nie twórz osobnego stanu na przefiltrowaną tablicę.

1. Utwórz komponent `Filtry` z formularzem kontrolowanym: pole wyszukiwania po nazwie (bez względu na wielkość liter), lista kategorii (zbudowana z danych przez `new Set`, plus opcja „wszystkie”), lista sortowania i pole wyboru „tylko dostępne”. Stany trzymaj w App i przekazuj przez props.
2. Oblicz tablicę `widoczne` łańcuchem `filter(...)` i sortowaniem kopii tablicy (`toSorted` albo `[...tab].sort`): nazwa A–Z (`localeCompare(b, 'pl')`), cena rosnąco, cena malejąco, najlepiej oceniane.
3. Wyświetl licznik „Wyświetlono X z Y gier” i komunikat, gdy żadna gra nie spełnia warunków. Zapisz w sprawozdaniu wynik dla: szukaj „ko”; kategoria rodzinna + tylko dostępne + cena rosnąco.

5 Wersja produkcyjna

1. W `vite.config.js` dodaj `base: './'`, uruchom `npm run build` i zapisz rozmiar plików JS i CSS z komunikatu Vite.
2. Uruchom `npm run preview` i sprawdź działanie wersji z folderu `dist`. Następnie skopiuj zawartość `dist` do `htdocs/katalog` i otwórz `http://localhost/katalog/`.
3. Otwórz plik `dist/index.html` bezpośrednio z dysku (dwuklik) i zanotuj, co widać w konsoli F12. Wyjaśnij, dlaczego wersję produkcyjną trzeba serwować przez serwer HTTP.
4. Dodaj do `.gitignore` foldery `node_modules` i `dist` (jeśli ich brak) i spakuj projekt bez `node_modules` do folderu `nazwisko_imie`.

W sprawozdaniu umieść

- ☐ zrzuty: stan ładowania, stan błędu, lista z filtrami w działaniu
- ☐ fragment kodu `useEffect` z obsługą błędu i `AbortController`
- ☐ wyniki filtrów z kroku 4 (liczba gier i kolejność)
- ☐ rozmiary plików z `npm run build` i obserwacja z otwarcia `dist` z dysku
- ☐ projekt bez `node_modules` i odpowiedzi na pytania

Pytania kontrolne

1. Co się stanie, jeśli w `useEffect` pominiesz tablicę zależności, a w efekcie ustawiasz stan?
2. Dlaczego `fetch` nie zgłasza wyjątku przy kodzie 500 i jak to obsłużyłeś?
3. Dlaczego listy przefiltrowanej nie trzeba przechowywać w osobnym stanie?
4. Po co nagłówek `Access-Control-Allow-Origin` w `api.php`, skoro Vite i Apache działają na tym samym komputerze?

Kryteria oceny

Kryterium	Punkty
Pobieranie danych w <code>useEffect</code> , sprawdzenie odp.ok, <code>AbortController</code>	4
Trzy stany interfejsu (ładowanie, błąd z przyciskiem ponowienia, dane) — zrzuty	3
Komponenty <code>GraKarta</code> i <code>Filtry</code> z props, <code>key</code> w liście, wyróżnienie gier niedostępnych	3
Filtrowanie (tekst, kategoria, dostępność) i 4 sposoby sortowania, licznik wyników	4
<code>Build</code> , <code>preview</code> i działanie z <code>htdocs</code> , <code>base './'</code> , <code>.gitignore</code>	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik `.docx` o tym samym numerze).

Git w zespole — gałęzie, pull requesty i konflikty

Czas: 2 × 45 min

praca w zespołach 3-osobowych (lider A, osoby B i C); przy 2 osobach lider wykonuje też zadania C

Cel: Praca zespołowa w jednym repozytorium GitHub: Issues jako lista zadań, gałęzie funkcji, pull requesty z przeglądem kodu, rozwiązywanie konfliktu scalania i oznaczenie wersji tagiem.

Po zajęciach potrafisz:

- planuje pracę zespołu w GitHub Issues i łączy zmiany z zadaniami (Closes #n)
- tworzy gałęzie funkcji, wysyła je na GitHub i otwiera pull request
- przegląda kod kolegi, komentuje konkretne linie i zatwierdza zmiany
- rozwiązuje konflikt scalania lokalnie i rozumie znaczniki <<<<<< ===== >>>>>>
- oznacza wersję tagiem z adnotacją i pisze README projektu

Sprzęt i materiały

- komputer z Windows 11, Git i VS Code
- konto GitHub dla każdej osoby (darmowe)
- folder `r2_24_start` z Pliki_do_zadan

Oprogramowanie

- **Git for Windows** — Git Bash i polecenia `git` w terminalu VS Code
git-scm.com/downloads
- **GitHub** — repozytorium, Issues, pull requesty, Releases
github.com
- **Visual Studio Code** — panel Source Control i edytor konfliktów (Accept Current / Incoming / Both)
code.visualstudio.com

Przygotowanie przed zajęciami

- Przypomnij sobie polecenia z zadania 28 roku 1 (commit, gałąź, merge, push).
- Sprawdź logowanie do GitHub z Git (np. `git push` do własnego repozytorium testowego); ustaw `git config --global user.name` i `user.email`.

Bezpieczeństwo

- Nie zapisuj haseł ani tokenów w plikach repozytorium; przy logowaniu używaj okna Git Credential Manager.
- Po zajęciach na komputerze szkolnym wyloguj się z GitHub i usuń zapisane poświadczenia (Menedżer poświadczeń Windows → Poświadczenia systemu Windows → `git:https://github.com`).

Przebieg zadania

1 Repozytorium zespołu i lista zadań (lider A)

Podział: #1 podstrona Projekty (B), #2 podstrona Kontakt (C), #3 styl aktywnego odnośnika w menu (C), #4 README (A).

1. Utwórz na GitHubie publiczne repozytorium `koło-programistów` (bez README). Skopiuj do lokalnego folderu pliki z `r2_24_start`, dodaj `.gitignore` (np. `node_modules/`, `.vscode/`, `Thumbs.db`), wykonaj pierwszy commit i `git push -u origin main`.
2. Dodaj członków zespołu: Settings → Collaborators → Add people. Każdy musi zaakceptować zaproszenie (e-mail lub github.com/notifications).
3. Utwórz Issues #1–#4 z opisem i przypisz osoby (Assignees). Dodaj etykiety `enhancement` lub `documentation`.
4. Włącz ochronę gałęzi main: Settings → Rules → Rulesets → New branch ruleset, cel: Default branch, reguła „Require a pull request before merging” z 1 wymaganą akceptacją, status Active.

2 Gałęzie funkcji (B i C równolegle)

B i C pracują jednocześnie, każde na swojej gałęzi. Oboje dopisują odnośnik w menu zaraz pod „Start” — konflikt jest częścią ćwiczenia.

```
git clone https://github.com/<lider>/koło-programistów.git
cd koło-programistów
git switch -c feature/projekty      # osoba C: feature/kontakt
# ... zmiany ...
git add .
git commit -m "Dodaj podstronę Projekty (#1)"
git push -u origin feature/projekty
```

1. B: utwórz `projekty.html` (ten sam nagłówek, menu i stopka, sekcja z 3 projektami koła) i dopisz w menu obu stron odnośnik „Projekty” bezpośrednio pod „Start”. Zrób co najmniej 2 commity.
2. C: utwórz `kontakt.html` z formularzem (imię, e-mail, wiadomość, przycisk) i dopisz w menu obu stron odnośnik „Kontakt” bezpośrednio pod „Start”. Zrób co najmniej 2 commity.
3. B i C: wyślij gałąź na GitHub i otwórz pull request do main. W opisie napisz, co zmieniono, i dodaj `Closes #1` (lub `Closes #2`).

3 Przegląd kodu i scalenie

Autor nie może zatwierdzić własnego pull requesta — robi to inna osoba z zespołu.

1. A: otwórz PR osoby B, karta Files changed. Dodaj komentarz do co najmniej jednej konkretnej linii (np. brak atrybutu alt, literówka), a następnie Review changes → Request changes lub Approve.
2. B: popraw kod zgodnie z uwagą, zrób commit i push na tę samą gałąź — sprawdź, że commit pojawił się w PR. Po akceptacji A scala PR (Merge pull request) i usuwa gałąź (Delete branch). Sprawdź, że Issue #1 zamknęło się automatycznie.
3. Zrób zrzut PR z komentarzem do linii i statusem Merged.

4 Konflikt scalania (C)

Po scaleniu PR #1 GitHub pokaże w PR osoby C: „This branch has conflicts that must be resolved”.

```
git switch feature/kontakt
git fetch origin
git merge origin/main          # CONFLICT (content): Merge conflict in index.html
# popraw pliki, usuń znaczniki konfliktu
git add index.html
git commit
git push
```

1. C: scal aktualny main do swojej gałęzi poleceniami z ramki. Otwórz `index.html` w VS Code i zrób zrzut znaczników konfliktu. Zapisz, co oznacza część HEAD, a co część origin/main.
2. Rozwiąż konflikt tak, aby menu miało kolejno: Start, Projekty, Kontakt (opcja Accept Both Changes i ewentualna poprawka ręczna). Sprawdź, że w żadnym pliku nie zostały znaczniki `<<<<<<` (`git grep "<<<<<<"`).
3. Zakończ scalanie commitem i wyślij gałąź. Po akceptacji B scal PR #2. Wykonaj w tym samym trybie Issue #3 (osobna gałąź i PR).

5 README, tag wersji i historia (A)

1. Na gałęzi `docs/readme` uzupełnij `README.md`: opis projektu, tabela zespołu (osoba, rola, Issues), uruchomienie, zasady pracy z gałęziami i PR, sekcja Wersje. Scal przez PR (Closes #4).
2. Wszyscy: `git switch main` i `git pull`. Lider: `git tag -a v1.0 -m "Wersja 1.0"` i `git push origin v1.0`, a następnie Releases → Draft a new release z tagiem v1.0 i krótkim opisem zmian.
3. Każdy: wykonaj `git log --oneline --graph --all` i zrób zrzut. Wskaż w historii commit scalający rozwiązujący konflikt.
4. Sprawdź w Insights → Contributors (lub Insights → Network), że każda osoba ma commity w repozytorium. Wpisz adres repozytorium w sprawozdaniu.

W sprawozdaniu umieść

- ☐ adres repozytorium zespołu i podział ról
- ☐ zrzuty: lista Issues, PR z komentarzem do linii, znaczniki konfliktu, historia `git log --graph`, Release v1.0
- ☐ opis rozwiązania konfliktu (co było po obu stronach i co zostawiono)
- ☐ odpowiedzi na pytania kontrolne (indywidualnie)

Pytania kontrolne

1. Dlaczego zespół pracuje na gałęziach funkcji zamiast commitować bezpośrednio do main?
2. Kiedy powstaje konflikt scalania, a kiedy Git scala zmiany automatycznie?
3. Czym różni się tag od gałęzi? Dlaczego `git push` nie wysyła tagów?
4. Co powinno się sprawdzać podczas przeglądu kodu (podaj 3 rzeczy)?

Kryteria oceny

Kryterium	Punkty
Repozytorium z .gitignore, współpracownicy, 4 Issues przypisane do osób, reguła ochrony main	3
Gałęzie funkcji z min. 2 commitami każdej osoby, PR z opisem i Closes #n	4
Przegląd kodu: komentarz do linii, poprawka w PR, scalenie, zamknięte Issue	3
Konflikt rozwiązany poprawnie (menu Start, Projekty, Kontakt, brak znaczników), zrzut i opis	4
README wg wzoru, tag v1.0 wysłany i Release, zrzut historii	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Bootstrap 5 — szybki interfejs sklepu

Czas: 2 × 45 min

praca indywidualna

Cel: Zbudowanie responsywnej strony sklepu z gotowych komponentów Bootstrap 5 (siatka, navbar, karty, formularz z walidacją, modal) oraz porównanie tego podejścia z własnym CSS i z instalacją przez npm.

Po zajęciach potrafisz:

- dołącza Bootstrap z CDN z atrybutem integrity i jako zależność npm
- projektuje układ siatkę container/row/col z punktami przerwania (sm, md, lg)
- używa komponentów navbar z menu zwijanym, card, modal i formularza z klasami walidacji
- stosuje klasy narzędziowe (odstęp, flex, kolory, display) i nadpisuje wygląd zmiennymi CSS --bs-*
- ocenia zalety i wady frameworka CSS w porównaniu z własnym arkuszem

Sprzęt i materiały

- komputer z Windows 11, VS Code (Live Server) i Node.js LTS
- folder `r2_25_start` z Pliki_do_zadan (treść `tresc.txt`, folder `img`, folder `wlasny_css`)

Oprogramowanie

- **Dokumentacja Bootstrap 5.3** — przykłady komponentów do kopiowania i dostosowania
getbootstrap.com/docs/5.3/getting-started/introduction
- **Visual Studio Code** — rozszerzenie Live Server
code.visualstudio.com
- **Node.js LTS** — część 5: npm i Vite
nodejs.org

Przygotowanie przed zajęciami

- Przejrzyj w prezentacji 2.16 siatkę Bootstrapa (12 kolumn, punkty przerwania) i listę komponentów.

Bezpieczeństwo

- Dołączając pliki z CDN, używaj adresów z dokumentacji z atrybutami integrity i crossorigin — przeglądarka odrzuci zmieniony plik.
- Instaluj pakiety npm tylko z oficjalnego rejestru i o sprawdzonej nazwie (bootstrap, @popperjs/core).

Przebieg zadania

1 Szablon i pasek nawigacji

Utwórz folder `nazwisko_imie/cdn` i skopiuj do niego folder `img`. Treści wszystkich elementów weź z pliku `tresc.txt`.

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css"
rel="stylesheet"
  integrity="sha384-sRIIL4kxILFvY47J16cr9ZwB07vP4J8+LH7qKQnuqkuIAvNWLzeN8tE5YBuJZqJLB"
crossorigin="anonymous">
...
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-FKyoEForCGlyvwx9Hj09JcYn3nv7wiPVLz7YYwJrWVcXK/BmnVDxM+D2scQbITxI"
crossorigin="anonymous"></script>
```

1. Utwórz `index.html` (`lang="pl"`, meta viewport) i dołącz CSS oraz JS Bootstrapa jak w ramce (skrypt na końcu body). Dołącz też własny plik `style.css` po CSS Bootstrapa.
2. Dodaj navbar z klasą `navbar-expand-lg`: logo i nazwa sklepu, 5 odnośników menu (pierwszy z klasą `active` i `aria-current="page"`), po prawej przycisk „Koszyk” z plakietką (badge) 0. Na wąskim ekranie menu ma się zwiijać pod przycisk-hamburger.
3. Ustaw kolor marki `#2f6b3a`: w `style.css` utwórz klasę `.bg-herbata` dla paska i stopki oraz klasę `.btn-herbata`, nadpisując zmienne `--bs-btn-bg`, `--bs-btn-hover-bg` i pokrewne (nie zmieniaj plików Bootstrapa).

2 Sekcja powitalna i siatka produktów

1. Dodaj sekcję powitalną: w `container` wiersz `row` z kolumną tekstu `col-lg-7` (nagłówek z klasą `display-5`, akapit `lead`, dwa przyciski) i kolumną grafiki `col-lg-5`, ukrytą poniżej lg klasami `d-none d-lg-block`.
2. Wyświetl 6 produktów w siatce `row row-cols-1 row-cols-sm-2 row-cols-lg-3 g-4` jako karty `card h-100`: grafika `card-img-top` z alt, plakietki kategorii i dostępności (zielona / żółta „ostatnie sztuki” / szara), nazwa, opis, parzenie, cena i przycisk.
3. Dopasuj karty klasami narzędziowymi: `d-flex flex-column` w treści karty i `mt-auto` w wierszu z ceną, tak aby ceny i przyciski były w jednej linii we wszystkich kartach. Dla produktu niedostępnego przycisk ma atrybut `disabled`.
4. Sprawdź układ w trybie urządzeń F12 (Ctrl+Shift+M) dla szerokości 390, 768 i 1280 px. Zapisz liczbę kolumn kart dla każdej szerokości i zrób zrzuty.

3 Formularz, modal i stopka

1. Dodaj formularz newslettera wg `tresc.txt` w kolumnie `col-md-10 col-lg-7` wyśrodkowanej `justify-content-center`: pola imię i e-mail obok siebie od sm (`col-sm-6`), lista, pole zgody `form-check`, przycisk na całą szerokość (`w-100`).
2. Włącz walidację Bootstrapa: formularz z klasą `needs-validation` i atrybutem `novalidate`, komunikaty `invalid-feedback`, a w pliku `app.js` przy wysyłce dodaj klasę `was-validated`, gdy `checkValidity()` zwraca `false`; przy poprawnych danych pokaż `alert alert-success`.
3. Dodaj modal „Koszyk” otwierany przyciskiem z paska (`data-bs-toggle="modal"`), z nagłówkiem, przyciskiem zamknięcia z `aria-label="Zamknij"` i stopką. Dodaj stopkę strony z danymi sklepu (`flex-column flex-md-row`).

4 Porównanie z własnym CSS

Folder `wlasny_css` zawiera ten sam fragment strony (menu i 3 karty) napisany bez frameworka.

1. Otwórz `wlasny_css/index.html` i swoją stronę. Porównaj: liczbę linii/bajtów własnego CSS, zachowanie menu na telefonie, rozmiar pobranych plików CSS i JS (F12 → Network, kolumna Transferred).
2. Zapisz w sprawozdaniu tabelę porównania i co najmniej 2 zalety i 2 wady Bootstrapa.

5 Wersja przez npm i Vite

```
npm create vite@latest lisc-npm -- --template vanilla
cd lisc-npm
npm install
npm install bootstrap @popperjs/core

// src/main.js
import 'bootstrap/dist/css/bootstrap.min.css'
import './style.css'
import * as bootstrap from 'bootstrap'
```

1. Utwórz projekt Vite poleceniami z ramki, przenieś do `index.html` treść strony bez znaczników CDN, grafiki umieść w folderze `public/img`, a `style.css` i kod walidacji w `src`.
2. Sprawdź w `package.json` wpis bootstrap i jego zakres wersji (`^5.3.8`). Wyjaśnij, co oznacza znak `^`.
3. Uruchom `npm run dev`, a potem `npm run build` i zapisz rozmiary plików CSS i JS w dist. Spakuj obie wersje (bez node_modules) w folderze `nazwisko_imie`.

W sprawozdaniu umieść

- ☐ zrzuty strony dla 390, 768 i 1280 px (w tym rozwinięte menu na telefonie), formularza z błędami walidacji i otwartego modala
- ☐ tabela porównania z wersją z własnym CSS oraz zalety i wady Bootstrapa
- ☐ rozmiary plików z `npm run build` i wyjaśnienie znaku `^`
- ☐ foldery cdn i npm bez node_modules, odpowiedzi na pytania

Pytania kontrolne

1. Co oznacza zapis `row-cols-1 row-cols-sm-2 row-cols-lg-3` i od jakich szerokości działa każda klasa?
2. Dlaczego własny plik CSS dołączasz po CSS Bootstrapa i dlaczego lepiej nadpisywać zmienne `--bs-*` niż edytować `bootstrap.min.css`?
3. Do czego służy atrybut `integrity` przy plikach z CDN?
4. Kiedy wybraćbyś wersję z CDN, a kiedy instalację przez npm?

Kryteria oceny

Kryterium	Punkty
Bootstrap z CDN z integrity, navbar zwijany z aktywnym odnośnikiem i koszykiem	3
Sekcja powitalna i siatka 6 kart: 1/2/3 kolumny, wyrównane ceny, plakietki, disabled	4
Formularz z walidacją (was-validated, komunikaty, alert sukcesu)	3
Modal koszyka i stopka; kolor marki przez własne klasy i zmienne --bs-btn-*	3
Porównanie z własnym CSS (tabela, zalety, wady)	2
Wersja npm + Vite: działa, build, rozmiary	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

WordPress — instalacja i konfiguracja

Czas: 2 × 45 min

praca indywidualna

Cel: Ręczna instalacja WordPressa na XAMPP z osobną bazą i użytkownikiem, konfiguracja witryny, utworzenie stron, wpisów i menu w motywie blokowym oraz poznanie ról użytkowników.

Po zajęciach potrafisz:

- tworzy bazę i osobnego użytkownika MariaDB z uprawnieniami tylko do tej bazy
- konfiguruje plik wp-config.php (dane bazy, klucze, prefiks tabel, tryb debugowania)
- ustawia witrynę: strona główna statyczna, bezpośrednie odnośniki, strefa czasowa
- tworzy strony, wpisy z kategoriami i obrazkami oraz menu w edytorze witryny
- przydziela role użytkownikom i sprawdza ich uprawnienia

Sprzęt i materiały

- komputer z Windows 11 i XAMPP (Apache, MariaDB, PHP 8.x)
- paczka WordPress (.zip) pobrana z pl.wordpress.org (lub z udziału szkolnego)
- folder `r2_26_tresci` z Pliki_do_zadan (tresci.txt, logo i grafiki)

Oprogramowanie

- **WordPress (polska wersja)** — aktualna wersja 7.x; plik .zip
pl.wordpress.org/download
- **XAMPP** — Apache + MariaDB + phpMyAdmin
apachefriends.org/download.html
- **Generator kluczy WordPress** — 8 wierszy `define()` do wp-config.php
api.wordpress.org/secret-key/1.1/salt
- **Visual Studio Code** — edycja wp-config.php
code.visualstudio.com

Przygotowanie przed zajęciami

- Przejrzyj w prezentacji 2.17 budowę WordPressa (pliki + baza) oraz różnicę między wpisem a stroną.
- Jeśli sieć szkolna jest wolna, pobierz paczkę WordPress wcześniej.

Bezpieczeństwo

- Nie nazywaj konta administratora „admin” i użyj silnego hasła (generator w instalatorze); hasła zapisz w sprawozdaniu tylko w postaci „***”.
- Instalacja jest lokalna — nie udostępniaj XAMPP w sieci szkolnej. Nie używaj konta root bazy w wp-config.php.
- Używaj adresów e-mail w domenie testowej (`hetman.example`), nie prawdziwych adresów kolegów.

Przebieg zadania

1 Baza danych i pliki

Każda aplikacja powinna mieć własną bazę i własnego użytkownika bazy z uprawnieniami tylko do niej.

1. Rozpakuj WordPress do `C:\xampp\htdocs\hetman` (w folderze mają być bezpośrednio wp-admin, wp-content, wp-includes, a nie podfolder wordpress).
2. W phpMyAdmin utwórz bazę `wp_hetman` z porównywaniem `utf8mb4_unicode_ci`, a w Konta użytkowników → Dodaj konto użytkownika utwórz konto `wp_hetman@localhost` z silnym hasłem i nadaj mu wszystkie uprawnienia tylko do bazy wp_hetman (karta Baza danych w uprawnieniach konta). Możesz to zrobić także poleceniami SQL — zapisz je w sprawozdaniu.
3. Skopiuj `wp-config-sample.php` jako `wp-config.php` i uzupełnij DB_NAME, DB_USER, DB_PASSWORD, DB_HOST. Zastąp 8 wierszy kluczy wierszami z generatora, zmień `$table_prefix` na `kh_`.
4. Włącz tryb deweloperski: `WP_DEBUG true`, `WP_DEBUG_LOG true`, `WP_DEBUG_DISPLAY false`. Zapisz, gdzie będzie zapisywany dziennik błędów.

2 Instalacja i ustawienia ogólne

1. Otwórz `http://localhost/hetman/` i dokończ instalację: tytuł „Klub Szachowy Hetman”, własna nazwa administratora, silne hasło, e-mail testowy. Zaloguj się do kokpitu (`http://localhost/hetman/wp-admin`).
2. W phpMyAdmin sprawdź, ile tabel powstało i jaki mają prefiks. Znajdź w tabeli `kh_options` wiersze `siteurl` i `blogname`.
3. Ustawienia → Ogólne: opis witryny z `tresci.txt`, strefa czasowa Warszawa, format daty d.m.Y. Ustawienia → Bezpośrednie odnośniki: „Nazwa wpisu”; sprawdź, że w folderze hetman powstał plik `.htaccess`, i otwórz jego treść.
4. Usuń przykładowy wpis „Witaj, świecie!” i przykładową stronę. Ustawienia → Dyskusja: włącz ręczne zatwierdzanie komentarzy.

3 Strony, wpisy i media

Treści wszystkich stron i wpisów są w `tresci.txt`.

1. Utwórz strony: Strona główna (bloki: nagłówek, akapit, przycisk do Kontaktu, kolumny z trzema zaletami), O klubie, Treningi (blok Tabela z 3 grupami), Kontakt i pustą stronę Aktualności.
2. Ustawienia → Czytanie: strona główna wyświetla stronę statyczną „Strona główna”, strona wpisów — „Aktualności”.
3. Utwórz kategorie Turnieje, Treningi, Ogłoszenia (Wpisy → Kategorie) i 3 wpisy z `tresci.txt`, każdy z kategorią i obrazkiem wyróżniającym. Jeden wpis zaplanuj na przyszłą datę i sprawdź jego status.
4. W Media sprawdź, jakie rozmiary obrazka utworzył WordPress po wgraniu pliku (folder `wp-content/uploads/rok/miesiac`).

4 Wygląd: edytor witryny

Motyw blokowy edytujesz w Wygląd → Edytor (szablony, części szablonu, style, nawigacja).

1. W części szablonu Nagłówek wstaw blok Logo witryny z plikiem `logo_hetman.png` i ustaw nawigację: Start, O klubie, Treningi, Aktualności, Kontakt.
2. W Style zmień paletę kolorów lub kolor tła i krój pisma nagłówek. W części szablonu Stopka wpisz adres i telefon klubu. Zapisz zmiany i sprawdź stronę w trybie prywatnym przeglądarki.

5 Użytkownicy i role

1. Dodaj użytkowników z treści.txt (Użytkownicy → Dodaj nowego użytkownika) z rolami Redaktor, Autor i Współpracownik.
2. W oknie prywatnym zaloguj się jako współpracownik i napisz wpis „Wyniki turnieju”. Sprawdź, czy możesz go opublikować i wgrać obrazek. Następnie zaloguj się jako autor i sprawdź, czy możesz edytować strony i cudze wpisy.
3. Zaloguj się jako redaktor i opublikuj wpis współpracownika. Uzupełnij w sprawozdaniu tabelę uprawnień ról.

W sprawozdaniu umieść

- ☐ polecenia SQL (lub zrzuty) tworzenia bazy i użytkownika, fragment wp-config.php bez haseł i kluczy
- ☐ zrzut listy tabel z prefiksem kh_ i treść pliku .htaccess
- ☐ zrzuty strony głównej, strony Treningi i listy Aktualności
- ☐ tabela uprawnień ról z wynikami testu
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Gdzie WordPress przechowuje treść wpisów, a gdzie wgrane obrazki? Co trzeba skopiować, by zrobić pełną kopię witryny?
2. Po co osobny użytkownik bazy zamiast konta root?
3. Czym różni się wpis od strony? Podaj po dwa przykłady treści dla każdego typu.
4. Do czego służą klucze i sole w wp-config.php i co się stanie po ich zmianie?

Kryteria oceny

Kryterium	Punkty
Baza utf8mb4 i osobny użytkownik z uprawnieniami tylko do niej	3
wp-config.php: dane bazy, klucze z generatora, prefiks kh_, tryb debugowania do pliku	3
Ustawienia: opis, strefa czasowa, odnośniki „Nazwa wpisu”, strona główna statyczna, moderacja komentarzy	3
Strony (tabela, kolumny, przycisk) i 3 wpisy z kategoriami i obrazkami	4
Logo, menu z 5 pozycjami, zmiana stylu i stopka w edytorze witryny	2
Trzy konta z rolami, test uprawnień i tabela ról	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

WordPress — wtyczki, bezpieczeństwo i kopie zapasowe

Czas: 2 × 45 min

praca indywidualna

Cel: Rozbudowa witryny WordPress o wtyczki (formularz, SEO, kopie zapasowe), zabezpieczenie instalacji wg listy kontrolnej oraz przeniesienie witryny w nowe miejsce z kopii bazy i plików.

Po zajęciach potrafisz:

- instaluje, konfiguruje i usuwa wtyczki, oceniając ich wiarygodność
- dodaje formularz kontaktowy i ustawienia SEO (opis meta, mapa witryny)
- wykonuje kopię zapasową bazy i plików oraz planuje harmonogram kopii
- zabezpiecza instalację: aktualizacje, konta, ograniczenie prób logowania, DISALLOW_FILE_EDIT, uprawnienia plików
- przenosi witrynę do nowej lokalizacji i poprawia adresy w bazie

Sprzęt i materiały

- komputer z XAMPP i działającą witryną `http://localhost/hetman` z zadania 26
- pendrive lub dysk sieciowy na kopię zapasową

Oprogramowanie

- **Katalog wtyczek WordPress** — instaluj z kokpitu: Wtyczki → Dodaj wtyczkę wordpress.org/plugins
- **XAMPP / phpMyAdmin** — eksport i import bazy [apachefriends.org/download.html](https://httpd.apache.org/download.html)
- **Visual Studio Code** — edycja wp-config.php code.visualstudio.com

Przygotowanie przed zajęciami

- Sprawdź, czy witryna z zadania 26 działa i czy pamiętasz hasło administratora.
- Przypomnij sobie z prezentacji 2.17 zasady bezpieczeństwa WordPressa i z prezentacji 2.11 listę OWASP.

Bezpieczeństwo

- Instaluj wtyczki tylko z oficjalnego katalogu — nigdy płatnych wtyczek „za darmo” z innych stron.
- Ograniczenie prób logowania sprawdzaj tylko na własnym koncie testowym i własnej instalacji.
- Kopia zapasowa zawiera bazę z danymi użytkowników (e-maile, skróty haseł) — przechowuj ją poza folderem htdocs i nie wysyłaj nikomu.

Przebieg zadania

1 Wybór i instalacja wtyczek

Przed instalacją sprawdź na karcie wtyczki: liczbę aktywnych instalacji, datę ostatniej aktualizacji, zgodność z Twoją wersją WordPressa i oceny.

1. Zapisz w tabeli dane 4 wtyczek: Contact Form 7, Yoast SEO (lub Rank Math), UpdraftPlus, Limit Login Attempts Reloaded — aktywne instalacje, ostatnia aktualizacja, zgodność. Zainstaluj je i włącz.
2. Contact Form 7: utwórz formularz „Zapis na trening” (imię, e-mail, grupa — lista Żaki/Juniorzy/Seniorzy, wiadomość, pole akceptacji) i wstaw jego shortcode na stronie Kontakt. Wyślij formularz i zapisz komunikat — wyjaśnij, dlaczego na XAMPP wiadomość nie dociera.
3. Yoast SEO: dla Strony głównej i Treningów ustaw frazę kluczową i opis meta (do 155 znaków). Otwórz mapę witryny `http://localhost/hetman/sitemap_index.xml` i zapisz, jakie mapy zawiera. W źródle strony (Ctrl+U) znajdź znacznik meta description i og:title.

2 Kopia zapasowa

Pełna kopia WordPressa = baza danych + folder wp-content (wtyczki, motywy, uploads) + wp-config.php.

1. UpdraftPlus (Ustawienia → Kopie zapasowe UpdraftPlus): wykonaj kopię ręczną bazy i plików. Sprawdź w `wp-content/updraft`, ile plików powstało i co zawierają, i skopiuj je pozahtdocs.
2. Ustaw harmonogram: pliki co tydzień, baza codziennie, przechowywane 4 kopie. Wyjaśnij, dlaczego kopie przechowywane tylko na tym samym serwerze nie wystarczą (zasada 3-2-1).
3. Wykonaj też kopię ręczną: phpMyAdmin → baza wp_hetman → Eksportuj → SQL. Zapisz rozmiar pliku.

3 Aktualizacje i zabezpieczenia

1. Kokpit → Aktualizacje: zainstaluj oczekujące aktualizacje (najpierw kopia!). Na liście wtyczek włącz automatyczne aktualizacje dla zainstalowanych wtyczek.
2. Narzędzia → Kondycja witryny: zapisz uwagi krytyczne i zalecane oraz popraw te, które się da (np. usuń nieużywane motywy i wtyczki, np. przykładową wtyczkę Hello Dolly, jeśli jest).
3. Dopisz do wp-config.php `define( 'DISALLOW_FILE_EDIT', true );` i sprawdź, że w menu Wygląd zniknęła Edytor plików motywu. Wyjaśnij, przed czym chroni to ustawienie.
4. Limit Login Attempts Reloaded: w oknie prywatnym zaloguj się 2 razy złym hasłem na własne konto testowe i zapisz komunikat. Sprawdź listę zdarzeń w ustawieniach wtyczki.
5. Przejrzyj Użytkownicy: tylko jedno konto administratora, silne hasła. Uzupełnij tabelę uprawnień plików dla hostingu Linux: katalogi, pliki, wp-config.php (wartości ósemkowe i znaczenie).

4 Migracja witryny w nowe miejsce

Symulujesz przeniesienie na inny serwer: nowa lokalizacja `http://localhost/hetman2` i nowa baza `wp_hetman2`.

```
UPDATE kh_options SET option_value = 'http://localhost/hetman2'
WHERE option_name IN ('siteurl', 'home');
```

1. Skopiuj folder `htdocs\hetman` jako `htdocs\hetman2`. Utwórz bazę `wp_hetman2`, nadaj użytkownikowi `wp_hetman` uprawnienia do niej i zaimportuj eksport SQL z kroku 2.
2. W `hetman2/wp-config.php` zmień `DB_NAME`, a w bazie `wp_hetman2` wykonaj zapytanie z ramki. Zaloguj się do `http://localhost/hetman2/wp-admin` i zapisz ponownie Ustawienia → Bezpośrednie odnośniki.
3. Zainstaluj w `hetman2` wtyczkę Better Search Replace i zamień `http://localhost/hetman/` na `http://localhost/hetman2/` we wszystkich tabelach — najpierw z opcją próbną (dry run), zapisz liczbę zmian, potem naprawdę. Wyjaśnij, dlaczego nie robi się tego zwykłym REPLACE w SQL.
4. Sprawdź nową witrynę: strony, wpisy, obrazki, menu, formularz. Wyłącz Apache-owy dostęp do starej wersji, zmieniając nazwę folderu `hetman`, i potwierdź, że `hetman2` nie odwołuje się do starych adresów (F12 → Network).

W sprawozdaniu umieść

- ☐ tabela oceny wtyczek i rzuty: formularz na stronie Kontakt, ustawienia Yoast, mapa witryny
- ☐ lista plików kopii UpdraftPlus, harmonogram i opis zasady 3-2-1
- ☐ uwagi z Kondycji witryny i wykonane poprawki, tabela uprawnień plików
- ☐ przebieg migracji z zapytaniem SQL i liczbą zmian Better Search Replace
- ☐ odpowiedzi na pytania kontrolne

Pytania kontrolne

1. Po czym poznasz, że wtyczce można zaufać? Dlaczego nieużywane wtyczki należy usuwać, a nie tylko wyłączać?
2. Co zawiera pełna kopia witryny WordPress i jak sprawdzić, że kopia jest użyteczna?
3. Dlaczego po przeniesieniu bazy trzeba zmienić `siteurl` i `home` oraz adresy w treści?
4. Jakie trzy ustawienia z dzisiejszych zajęć najbardziej zmniejszają ryzyko przejścia witryny i dlaczego?

Kryteria oceny

Kryterium	Punkty
Tabela oceny 4 wtyczek, instalacja i konfiguracja	2
Formularz Contact Form 7 na stronie Kontakt, wyjaśnienie poczty na XAMPP	2
SEO: frazy i opisy meta, mapa witryny, znaczniki w źródle	2
Kopia UpdraftPlus i eksport SQL, harmonogram, zasada 3-2-1	3
Aktualizacje, Kondycja witryny, <code>DISALLOW_FILE_EDIT</code> , limit logowań, uprawnienia plików	4
Migracja: działająca witryna <code>hetman2</code> , <code>siteurl/home</code> , Better Search Replace	4
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Wydajność i dostępność w praktyce

Czas: 2 × 45 min

praca indywidualna

Cel: Zmierzenie wydajności i dostępności strony narzędziem Lighthouse, poprawienie jej (obrazy WebP/AVIF, lazy loading, defer, cache, poprawki WCAG 2.2) i udokumentowanie efektu pomiarem przed i po.

Po zajęciach potrafisz:

- przeprowadza audyt Lighthouse i interpretuje wskaźniki Core Web Vitals (LCP, INP, CLS)
- optymalizuje obrazy: format WebP/AVIF, właściwy rozmiar, srcset, wymiary, lazy loading
- usuwa zasoby blokujące renderowanie i konfiguruje cache przeglądarki w .htaccess
- poprawia dostępność wg WCAG 2.2: tekst alternatywny, kontrast, etykiety, nagłówki, fokus, język strony
- dokumentuje wyniki pomiarów przed i po zmianach

Sprzęt i materiały

- komputer z XAMPP (Apache), Chrome lub Edge, VS Code i GIMP
- folder `r2_28_start` z Pliki_do_zadan

Oprogramowanie

- **Lighthouse w Chrome/Edge** — F12 → karta Lighthouse
developer.chrome.com/docs/lighthouse/overview
- **Squoosh** — konwersja do WebP/AVIF w przeglądarce (lub GIMP: Plik → Wyeksportuj jako .webp)
squoosh.app
- **WebAIM Contrast Checker** — kontrast tekstu (AA: 4,5:1)
webaim.org/resources/contrastchecker
- **XAMPP** — Apache obsługuje plik .htaccess
httpd.apache.org/download.html

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 2.18 wskaźniki Core Web Vitals i z roku 1 zasady dostępności (alt, etykiety, kontrast).

Bezpieczeństwo

- Audyt uruchamiaj w oknie InPrivate/incognito — rozszerzenia przeglądarki zaniżają wynik i mogą mieć dostęp do strony.
- Przed zmianami zrób kopię folderu `r2_28_start` (porównasz wersje i zawsze możesz wrócić).

Przebieg zadania

1 Pomiar wyjściowy

Core Web Vitals: LCP (największy element widoczny) $\leq 2,5$ s, INP (reakcja na interakcję) ≤ 200 ms, CLS (przesunięcia układu) $\leq 0,1$. Lighthouse mierzy w laboratorium LCP, CLS i TBT (zamiast INP).

1. Skopiuj `r2_28_start` do `C:\xampp\htdocs\polana` i otwórz `http://localhost/polana/` w oknie InPrivate.
2. F12 → Lighthouse: tryb Navigation, urządzenie Mobile, kategorie Performance, Accessibility, Best practices, SEO. Uruchom audyt i zapisz w tabeli 4 wyniki oraz FCP, LCP, TBT, CLS i całkowity rozmiar strony. Zapisz raport (menu : → Save as HTML).
3. F12 → Network (wyłącz cache, odśwież): zapisz liczbę żądań i rozmiar przesłanych danych. Wypisz 6 najważniejszych uwag z raportu.

2 Obrazy

Zdjęcie w galerii wyświetla się w szerokości ok. 480 px — plik 1920 px i 3 MB to marnotrawstwo.

1. Przygotuj obrazy w Squoosh lub GIMP: tło sekcji powitalnej w szerokościach 800 i 1600 px, zdjęcia galerii 400 i 800 px — każde w WebP i AVIF. Zapisz w tabeli rozmiar przed i po dla jednego zdjęcia.
2. Zastąp tło CSS sekcji powitalnej elementem `<picture>` z `<source type="image/avif">`, `<source type="image/webp">` i `srcset / sizes`; obrazek wyświetl przez `object-fit: cover`. Dodaj mu `fetchpriority="high"` (to element LCP — nie może mieć lazy loading).
3. W galerii dodaj `srcset`, `sizes`, `width`, `height`, `loading="lazy"` i `decoding="async"`. W CSS ustaw `height: auto`. Wyjaśnij, jak width/height chronią przed przesunięciami (CLS).

3 Skrypty, cache i kompresja

```
<IfModule mod_expires.c>
  ExpiresActive On
  ExpiresByType image/webp "access plus 1 year"
  ExpiresByType text/css "access plus 1 month"
  ...
</IfModule>
```

1. Przenieś skrypt z head na końcowy `defer`, usuń sztuczną pętlę obliczeń, a przycisk-div zastąp prawdziwym `<button type="submit">` z obsługą zdarzenia submit.
2. Utwórz plik `.htaccess` z regułami `mod_expires` (obrazy 1 rok, CSS/JS 1 miesiąc, HTML 0 s) i `mod_deflate` dla HTML, CSS, JS i SVG. Sprawdź w F12 → Network → nagłówki odpowiedzi `Expires / Cache-Control` i `Content-Encoding` dla pliku CSS.
3. Wyjaśnij, dlaczego HTML ma krótki czas cache, a obrazy długi, i jak wtedy wprowadzić nową wersję obrazka (zmiana nazwy pliku).

4 Dostępność (WCAG 2.2) i SEO

1. Dodaj `lang="pl"`, meta viewport, opisowy title i meta description. Zamień div-y na `header`, `nav` (z `aria-label`), `main`, `footer`; nagłówki sekcji zmień z h4 na h2.
2. Dodaj teksty alternatywne opisujące zdjęcia (obraz dekoracyjny tła: `alt=""`), zmień link „Kliknij tutaj” na opisowy, dodaj `<label for>` do każdego pola, typy pól email i date oraz autocomplete.
3. Popraw kontrast: tekst szary #aaa i jasne odnośniki w menu i stopce zmień tak, by miały co najmniej 4,5:1 (sprawdź w WebAIM). Usuń `outline: none` i dodaj wyraźny styl `:focus-visible`; przejdź stronę klawiszem Tab i sprawdź, że widać fokus.
4. Dodaj odnośnik „Przejdź do treści” widoczny po fokusie (WCAG: pomijanie bloków).

5 Pomiar końcowy

1. Uruchom ponownie audyt Lighthouse (te same ustawienia) i uzupełnij kolumnę „po” w tabeli. Zapisz raport HTML.
2. Sprawdź stronę narzędziem do podglądu dostępności (F12 → Elements → Accessibility lub F12 → Lighthouse) i wymień jeden problem, którego automatyczny audyt nie wykrywa (np. sens tekstu alt, kolejność czytania).
3. Umieść wersję przed i po oraz raporty w folderze `nazwisko_imie`.

W sprawozdaniu umieść

- ☐ tabela wyników Lighthouse i Network przed i po (4 kategorie, FCP, LCP, TBT, CLS, rozmiar, liczba żądań)
- ☐ rozmiary jednego zdjęcia: PNG, WebP, AVIF w dwóch szerokościach
- ☐ fragmenty kodu: picture z srcset, .htaccess, poprawiony formularz
- ☐ lista poprawek dostępności z odniesieniem do zasad WCAG
- ☐ raporty HTML Lighthouse i odpowiedzi na pytania

Pytania kontrolne

1. Co mierzą LCP, INP i CLS i jakie są progi „dobrego” wyniku?
2. Dlaczego obrazu LCP nie wolno ładować z loading="lazy"?
3. Czym różni się atrybut defer od async przy skryptach?
4. Dlaczego wynik 100 w kategorii Accessibility nie oznacza, że strona jest w pełni dostępna?

Kryteria oceny

Kryterium	Punkty
Pomiar wyjściowy: tabela 4 kategorii i metryk, raport HTML, 6 uwag	3
Obrazy WebP/AVIF w 2 szerokościach, picture + srcset, width/height, lazy, fetchpriority	4
Skrypt defer, button zamiast div, .htaccess z cache i kompresją sprawdzony w nagłówkach	3
Poprawki dostępności: lang, struktura, alt, etykiety, kontrast $\geq 4,5:1$, focus-visible, skip link	4
Pomiar końcowy i porównanie przed/po	3
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Wdrożenie aplikacji PHP + MySQL

Czas: 2 × 45 min

praca indywidualna

Cel: Przygotowanie aplikacji PHP + MySQL do wdrożenia (konfiguracja poza kodem, katalog public, bezpieczne błędy, konto bazy z minimalnymi uprawnieniami) i przeniesienie jej na serwer przez SFTP z testem po wdrożeniu; Docker Compose jako alternatywa lokalna.

Po zajęciach potrafisz:

- przenosi dane dostępowe do pliku `.env` i zmiennych środowiskowych, a do repozytorium dodaje `.env.example`
- rozdziela katalog publiczny od kodu i konfiguracji oraz blokuje dostęp do plików wrażliwych
- konfiguruje obsługę błędów dla produkcji (`display_errors Off`, dziennik błędów)
- eksportuje i importuje bazę, tworzy konto aplikacji z minimalnymi uprawnieniami
- wysyła pliki przez SFTP, sprawdza HTTPS i wykonuje test po wdrożeniu wg listy kontrolnej

Sprzęt i materiały

- komputer z XAMPP i VS Code
- konto na serwerze szkolnym lub darmowym hostingu z PHP 8, MySQL/MariaDB i SFTP (dane od nauczyciela)
- folder `r2_29_start` z Pliki_do_zadan (aplikacja i plik `usterki.sql`)

Oprogramowanie

- **FileZilla Client** — protokół SFTP; alternatywa: WinSCP <https://winscp.net/filezilla-project.org/download.php?type=client>
- **XAMPP** — wersja deweloperska i eksport bazy apachefriends.org/download.html
- **Docker Desktop (opcjonalnie)** — krok 5 — jeśli jest zainstalowany w pracowni docker.com/products/docker-desktop

Przygotowanie przed zajęciami

- Przypomnij sobie z prezentacji 2.18 środowiska dev/test/prod oraz z zadania 29 roku 1 pracę z FileZillą.
- Odbierz od nauczyciela dane: adres serwera, login SFTP, nazwę bazy i hasło.

Bezpieczeństwo

- Hasła do serwera i bazy trzymaj tylko w pliku `.env` na serwerze i w menedżerze haseł — nigdy w repozytorium ani w sprawozdaniu (pisz „***”).
- Używaj SFTP (szyfrowane), nie zwykłego FTP. Nie zapisuj hasła w FileZilli na komputerze szkolnym.
- Plik z `phpinfo()` usuń zaraz po sprawdzeniu ustawień — ujawnia konfigurację serwera.

Przebieg zadania

1 Uruchomienie lokalne i przegląd problemów

1. Zaimportuj `usterki.sql` w phpMyAdmin, skopiuj aplikację do `htdocs\usterki` i sprawdź działanie (lista 5 zgłoszeń, dodanie nowego).
2. Zmień w `db.php` hasło na błędne i zrób zrzut komunikatu. Wypisz, jakie informacje o serwerze zdradza ten komunikat, oraz 4 inne problemy aplikacji przed wdrożeniem (np. konto root, hasło w kodzie, dostęp do pliku `.sql` z przeglądarki).

2 Konfiguracja przez zmienne środowiskowe

Docelowa struktura: `public/` (`index.php`, `style.css` — jedyny katalog widoczny z WWW), `src/` (`config.php`, `db.php`), `sql/`, `logs/`, plik `.env`, `.env.example`, `.gitignore`.

```
# .env.example
APP_ENV=development
DB_HOST=localhost
DB_PORT=3306
DB_NAME=usterki
DB_USER=usterki_app
DB_PASS=
```

1. Przebuduj strukturę katalogów jak w opisie i popraw ścieżkę w `public/index.php` (`require __DIR__ . '/../src/db.php'`).
2. Napisz `src/config.php`: funkcja wczytująca plik `.env` (wiersze KLUCZ=wartość, pomijanie komentarzy #) przez `putenv`, funkcja `env($klucz, $domyslna)` oparta na `getenv`. Zmienne ustawione przez serwer mają pierwszeństwo przed plikiem.
3. W `config.php` ustaw obsługę błędów zależnie od `APP_ENV`: w production `display_errors 0`, zawsze `log_errors 1` i `error_log` do `logs/php-error.log`.
4. W `src/db.php` zbuduj DSN z wartości env, a wyjątek połączenia złap: zapisz szczegóły `error_log`, zwróć kod 503 i ogólny komunikat. Sprawdź oba tryby (development/production) z błędnym hasłem.
5. Utwórz konto bazy `usterki_app` z uprawnieniami tylko SELECT, INSERT, UPDATE, DELETE do bazy `usterki`. Dodaj `.gitignore` z `.env` i `logs/*.log`.

3 Wdrożenie na serwer

Jeśli katalogu głównego domeny nie da się ustawić na `public/`, dodaj w katalogu aplikacji `.htaccess` z przekierowaniem do `public/` i blokadą `src/`, `sql/`, `logs/` i `.env`.

1. Wyeksportuj bazę z lokalnego phpMyAdmin (Eksportuj → SQL) i zaimportuj ją w phpMyAdmin serwera. Na serwerze konto bazy tworzy zwykle panel hostingu — zapisz, jakie uprawnienia ma.
2. Połącz się FileZillą przez SFTP (port 22) i wyślij pliki bez `.env`. Na serwerze utwórz `.env` z danymi produkcyjnymi i `APP_ENV=production`.
3. Sprawdź na chwilę plikiem `info.php` z `phpinfo()` wartości `display_errors`, `log_errors` i wersję PHP, po czym usuń plik.

4 Test po wdrożeniu

1. Wykonaj test wg listy kontrolnej: strona przez https (kłódka, przekierowanie z http), lista z bazy, dodanie zgłoszenia, polskie znaki, odświeżenie po dodaniu (PRG).
2. Sprawdź, że adresy `/.env`, `/src/db.php`, `/sql/usterki.sql` zwracają 403 lub 404. Wynik każdego punktu zapisz w tabeli (OK / błąd i poprawka).

5 Docker Compose — alternatywa lokalna (jeśli dostępny)

Bez Dockera opisz tylko rolę każdej sekcji pliku docker-compose.yml.

```
services:
  app:
    build: . # Dockerfile: php:8.3-apache + pdo_mysql
    ports: ["8080:80"]
    environment: { DB_HOST: db, DB_NAME: usterki, DB_USER: usterki_app, DB_PASS: ... }
  db:
    image: mariadb:10.11
    environment: { MARIADB_DATABASE: usterki, MARIADB_USER: usterki_app, ... }
    volumes:
      - dane_db:/var/lib/mysql
      - ./sql/usterki.sql:/docker-entrypoint-initdb.d/01-usterki.sql:ro
```

1. Napisz Dockerfile (obraz php:8.3-apache, `docker-php-ext-install pdo_mysql`, katalog główny Apache = public) i docker-compose.yml wg ramki. Uruchom `docker compose up -d` i otwórz `http://localhost:8080`.
2. Wyjaśnij, dlaczego DB_HOST ma wartość `db`, a nie `localhost`, i co daje wolumen `dane_db`.

W sprawozdaniu umieść

- ☐ lista problemów wersji startowej z uzasadnieniem
- ☐ kod `config.php`, `db.php`, `.env.example`, `.gitignore` (bez haseł)
- ☐ adres wdrożonej aplikacji, zrzut z kłódką HTTPS i tabela testu po wdrożeniu
- ☐ zrzuty: komunikat 503 w produkcji i wpis w dzienniku błędów
- ☐ docker-compose.yml lub jego opis, odpowiedzi na pytania

Pytania kontrolne

1. Dlaczego dane dostępne do bazy nie powinny być zapisane w kodzie w repozytorium?
2. Czym różni się konfiguracja błędów PHP w środowisku deweloperskim i produkcyjnym?
3. Po co aplikacji osobne konto bazy z ograniczonymi uprawnieniami?
4. Dlaczego katalogiem głównym serwera powinien być public/, a nie cały projekt?

Kryteria oceny

Kryterium	Punkty
Analiza wersji startowej: zrzut błędu i 5 problemów	2
Struktura public/src, config.php z .env i getenv, .env.example, .gitignore	4
Obsługa błędów: display_errors zależne od APP_ENV, log do pliku, 503 przy braku bazy	3
Konto bazy z minimalnymi uprawnieniami, eksport i import bazy	2
Wdrożenie przez SFTP, .env na serwerze, HTTPS, test wg listy kontrolnej, blokada plików wrażliwych	4
Docker Compose (uruchomienie lub poprawny opis)	2
Odpowiedzi na pytania kontrolne	3
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Próba INF.03: pełny arkusz nr 2 — szkółka roślin

Czas: 2 × 45 min

praca samodzielna, warunki egzaminacyjne (bez internetu i notatek)

Cel: Samodzielne wykonanie w 90 minut kompletnego zadania egzaminacyjnego: baza danych z zapytaniami, przygotowanie grafiki, witryna HTML/CSS i skrypt PHP z mysql, a następnie samoocena według kryteriów i analiza błędów.

Po zajęciach potrafisz:

- importuje bazę i zapisuje 4 zapytania SQL (SELECT z warunkiem, JOIN, GROUP BY, ALTER TABLE) wraz ze zrzutami
- skaluje i konwertuje grafikę zgodnie z poleceniem
- buduje witrynę z banerem, dwoma blokami i stopką dokładnie wg specyfikacji
- pisze skrypt PHP z mysql, który generuje formularz i wyniki z bazy
- ocenia swoją pracę wg kryteriów egzaminacyjnych i planuje poprawę słabych obszarów

Sprzęt i materiały

- komputer z XAMPP, edytorem kodu (VS Code lub Notepad++), GIMP i przeglądarką
- folder `r2_30_arkusz` z Pliki_do_zadan:
`szkolka.sql`, `ogrod.png`, `tekst.txt`

Oprogramowanie

- **XAMPP** — Apache, MariaDB, phpMyAdmin
apachefriends.org/download.html
- **GIMP** — Obraz → Skaluj obraz; Plik → Wyeksportuj jako
gimp.org/downloads
- **Visual Studio Code lub Notepad++** — na egzaminie edytor bez odpowiedzi AI
code.visualstudio.com

Przygotowanie przed zajęciami

- Przejrzyj kryteria oceny arkuszy INF.03 i swoje błędy z zadania 19 (pierwszej próby).
- Na pulpicie utwórz folder o nazwie swojego nazwiska i imienia — wszystkie wyniki zapisuj w nim.

Bezpieczeństwo

- Pracuj samodzielnie, bez internetu i asystentów AI — wynik ma pokazać Twój rzeczywisty poziom przed egzaminem.
- Co kilka minut zapisuj pliki (Ctrl+S); na egzaminie nie ma możliwości odzyskania utraconej pracy.

Przebieg zadania

1 Część 1. Baza danych (ok. 20 min)

Tabela `rodzaje`: id, nazwa. Tabela `rosliny`: id, nazwa, rodzaj_id (klucz obcy do `rodzaje`), cena, wysokosc_cm, kwitnienie, dostepne (liczba sztuk w magazynie).

1. Zaimportuj plik `szkolka.sql` (tworzy bazę `szkolka`). Zrób zrzut struktury bazy w phpMyAdmin (np. Widok relacyjny / Projektant) i zapisz jako `baza.png`.
2. Zapisz w pliku `kwerendy.txt` i wykonaj zapytania; zrzut wyniku każdego zapisz jako `kw1.png` ... `kw4.png`.
3. Zapytanie 1: wybiera nazwę i cenę roślin o rodzaju z id 2, posortowane malejąco według ceny.
4. Zapytanie 2: wybiera nazwę rośliny, nazwę jej rodzaju (z tabeli `rodzaje`) i cenę dla roślin dostępnych w magazynie (`dostepne` większe od 0) i tańszych niż 20 zł.
5. Zapytanie 3: dla każdego rodzaju wyświetla jego nazwę, liczbę roślin i średnią cenę zaokrągloną do 2 miejsc, z aliasami `liczba` i `srednia_cena`.
6. Zapytanie 4: dodaje do tabeli `rosliny` kolumnę `promocja` typu logicznego z wartością domyślną `falsz`, niepustą.

2 Część 2. Grafika (ok. 5 min)

1. Plik `ogrod.png` przeskaluj do szerokości 400 px z zachowaniem proporcji i zapisz w formacie JPEG jako `ogrod.jpg` (jakość 80).

3 Część 3. Witryna i styl (ok. 35 min)

Plik `szkolka.php`, tytuł karty „Szkółka roślin”, język polski, kodowanie UTF-8, arkusz `styl.css`. Teksty weź z pliku `tekst.txt`. Układ: baner na całą szerokość, pod nim blok lewy (30%) i blok prawy (70%), na dole stopka.

1. Baner: obraz `ogrod.jpg` z tekstem alternatywnym „Ogród z kwitnącymi roślinami”, nagłówek h1 z `tekst.txt`.
2. Blok lewy: nagłówek h2 i lista punktowana 5 odnośników z `tekst.txt`; regulamin otwiera się w nowej karcie.
3. Blok prawy: nagłówek h2, akapit z `tekst.txt` i efekt skryptu (część 4). Stopka: akapit „Stronę wykonał: ” i Twoje imię i nazwisko.
4. Styl: krój Verdana dla wszystkich elementów; baner tło `#2E7D32`, biały tekst, wyśrodkowany; obraz w banerze pływa do lewej, wysokość 120 px, zaokrąglenie narożników 10 px; h1 rozmiar 40 px.
5. Styl: blok lewy tło `#A5D6A7`, prawy `#E8F5E9`, oba wysokość 600 px i marginesy wewnętrzne lewy i prawy 20 px; stopka tło `#1B5E20`, biały tekst, wyrównanie do prawej, marginesy wewnętrzne 20 px, czyszczenie opływania.
6. Styl: odnośniki pogrubione w kolorze `#1B5E20`, po najechaniu `#E65100`; blok wyniku `.roslina` — białe tło, lewe obramowanie 6 px ciągłe `#2E7D32`, marginesy zewnętrzne 10 px u góry i u dołu.

4 Część 4. Skrypt PHP (ok. 25 min)

Skrypt w bloku prawym łączy się z bazą `szkołka` (localhost, root, bez hasła), na końcu zamyka połączenie.

1. Skrypt 1: wypisuje formularz wysyłany metodą GET do `szkołka.php` z listą rozwijaną `rodzaj` — opcje generowane z tabeli `rodzaje` (wartość = id, tekst = nazwa) — i przyciskiem „Pokaż”.
2. Skrypt 2: jeśli formularz został wysłany, wybiera z bazy nazwę, cenę i okres kwitnienia roślin wybranego rodzaju dostępnych w magazynie, posortowane alfabetycznie. Każdą roślinę wypisuje w bloku `div` klasy `roslina`: nagłówek `h3` z nazwą i akapit „Cena: X zł, kwitnie: Y”. Wartość z formularza przekaz bezpieczniej (rzutowanie na `int` lub `mysqli_prepare`).
3. Gdy rodzaj nie ma dostępnych roślin, skrypt wypisuje akapit „Brak dostępnych roślin tego rodzaju”. Sprawdź działanie dla krzewów (Bez lilak ma 0 sztuk) i zrób zrzut strony `witryna.png`.

5 Samoocena i analiza błędów (po czasie egzaminu)

1. Po 90 minutach przestań pisać. Otrzymasz kryteria oceniania — oceń swoją pracę punkt po punkcie i wpisz wynik w tabeli samooceny.
2. Porównaj wynik z zadaniem 19: wypisz 3 błędy lub braki, które powtórzyły się, i plan ich poprawy przed egzaminem.

W sprawozdaniu umieść

- ☐ folder z plikami: `kwerendy.txt`, `baza.png`, `kw1.png` – `kw4.png`, `ogrod.jpg`, `szkołka.php`, `styl.css`, `witryna.png`
- ☐ tabela samooceny wg kryteriów i porównanie z zadaniem 19
- ☐ lista 3 powtarzających się błędów z planem poprawy

Pytania kontrolne

1. Dlaczego w zapytaniu 2 potrzebne jest złączenie tabel i co by się stało bez warunku ON?
2. Czym różni się `mysqli_fetch_assoc` od `mysqli_fetch_row`? Którą funkcję wybrałeś i dlaczego?
3. Dlaczego wartości z `$_GET` nie wolno wklejać bezpośrednio do zapytania SQL?
4. Która część arkusza zajęła Ci najwięcej czasu i jak to zmienisz na egzaminie?

Kryteria oceny

Kryterium	Punkty
Zapytania 1–4 poprawne, zapisane w <code>kwerendy.txt</code> , zrzuty <code>kw1</code> – <code>kw4</code> i <code>baza.png</code>	5
Grafika: 400 px szerokości, proporcje, JPEG	1
Struktura HTML: tytuł, lang, baner, bloki, lista z 5 odnośnikami (<code>target_blank</code>), stopka	4
Styl CSS zgodny ze specyfikacją (kolory, układ, rozmiary, hover, klasa <code>roslina</code>)	4
Skrypt 1: połączenie, lista opcji z bazy, formularz GET, zamknięcie połączenia	2
Skrypt 2: zapytanie z warunkiem i sortowaniem, bezpieczny parametr, wynik w <code>div.roslina</code> , komunikat o braku	2
Samoocena i analiza błędów	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik `.docx` o tym samym numerze).

Projekt końcowy cz. 1 — specyfikacja, baza i repozytorium

Czas: 2 × 45 min

praca w zespołach 2–3-osobowych

Cel: Zaplanowanie zespołowej aplikacji webowej: wymagania i przypadki użycia, model danych (ERD i skrypt SQL z danymi testowymi), makiety oraz repozytorium z podziałem zadań.

Po zajęciach potrafisz:

- formułuje wymagania funkcjonalne i нефункционалне oraz ustala ich priorytety metodą MoSCoW
- opisuje przypadki użycia ze scenariuszem głównym i alternatywnym
- projektuje bazę z kluczami obcymi i ograniczeniami oraz pisze skrypt SQL z danymi testowymi
- przygotowuje makiety ekranów dla telefonu i komputera
- zakłada repozytorium zespołu z Issues, kamieniami milowymi i podziałem pracy

Sprzęt i materiały

- komputer z XAMPP, VS Code i Git; konto GitHub każdej osoby
- folder `r2_31_projekt` z Pliki_do_zadan (tematy.txt, szablon_specyfikacji.md i listy kontrolne)

Oprogramowanie

- **draw.io (diagrams.net)** — ERD i makiety; alternatywa: phpMyAdmin → Projektant app.diagrams.net
- **Penpot (opcjonalnie)** — darmowe makiety w przeglądarce penpot.app
- **XAMPP / phpMyAdmin** — test skryptu SQL apacheFriends.org/download.html
- **GitHub** — Issues, Milestones, Projects github.com

Przygotowanie przed zajęciami

- W zespole wybierzcie 2 tematy z pliku `tematy.txt` (kolejność preferencji) — nauczyciel przydziela tematy bez powtórzeń.
- Przypomnij sobie pracę z gałęziami i pull requestami (zadanie 24) oraz relacje i klucze obce (przedmiot bazy danych).

Bezpieczeństwo

- Dane testowe mają być fikcyjne — nie wpisuj prawdziwych danych kolegów (imion z nazwiskami, telefonów, e-maili).
- Do repozytorium nie dodawaj haseł; w danych testowych zapisuj tylko skróty haseł z `password_hash`.

Przebieg zadania

1 Temat i wymagania

Wymagania minimalne (tematy.txt): min. 3 powiązane tabele z tabelą użytkowników, 2 role, CRUD głównego zasobu, co najmniej jedna reguła biznesowa, jeden endpoint JSON, responsywny interfejs.

1. Skopiuj `szablon_specyfikacji.md` jako `docs/specyfikacja.md`. Opisz cel aplikacji i aktorów (role) z ich uprawnieniami.
2. Spisz min. 8 wymagań funkcjonalnych z priorytetami MoSCoW (co najmniej 5 Must) i min. 4 wymagania нефункционаłne (środowisko, responsywność, dostępność, bezpieczeństwo).
3. Opisz min. 3 przypadki użycia: aktor, warunek wstępny, kroki scenariusza głównego i co najmniej jeden scenariusz alternatywny (np. błąd walidacji, kolizja terminu).

2 Model danych

1. Narysuj ERD w draw.io: encje, klucze główne i obce, typy relacji (1-N, N-M rozbite tabelą pośrednią). Zapisz jako `docs/erd.png`.
2. Napisz skrypt `sql/baza.sql`: SET NAMES utf8mb4, CREATE DATABASE z utf8mb4_polish_ci, tabele InnoDB z PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, ENUM lub CHECK tam, gdzie mają sens, indeks pod najczęstsze wyszukiwanie.
3. Dodaj dane testowe: min. 2 konta (po jednym na rolę, hasła jako skróty z `php -r "echo password_hash('...', PASSWORD_DEFAULT);"`) i min. 8 rekordów głównego zasobu. Uruchom skrypt w phpMyAdmin bez błędów.
4. Napisz i sprawdź 3 zapytania, których aplikacja będzie potrzebować (np. lista z JOIN, sprawdzenie reguły biznesowej, podsumowanie dla administratora). Zapisz je w `sql/zapytania.sql` z wynikami w komentarzach.

3 Makiety

Makieta to szkic układu (prostokąty, podpisy), nie gotowa grafika.

1. Przygotuj makiety min. 3 ekranów: lista głównego zasobu, formularz (dodawanie / rezerwacja), panel administratora — każdy w wersji telefon (360 px) i komputer.
2. Zaznacz na makietach elementy nawigacji i komunikaty (błąd walidacji, pusta lista). Zapisz w `docs/makiety/`.

4 Repozytorium i plan pracy

1. Utwórz repozytorium zespołu, dodaj współpracowników, `.gitignore` (np. `.env`, `node_modules/`, `vendor/`), `.env.example` i strukturę katalogów `public/`, `src/`, `sql/`, `docs/`.
2. Utwórz kamienie milowe (Issues → Milestones) „cz. 2 backend”, „cz. 3 frontend”, „cz. 4 wdrożenie” i min. 10 Issues z wymagań Must/Should, przypisanych do osób i kamieni milowych. Opcjonalnie: tablica Projects (To do / In progress / Done).
3. Uzupełnij w specyfikacji tabelę podziału pracy. Każda osoba wykonuje min. 1 commit przez pull request z przeglądem drugiej osoby.
4. Przedstaw nauczycielowi specyfikację i ERD (2 min na zespół) i zapisz uwagi jako Issues.

W sprawozdaniu umieść

- ☐ adres repozytorium i skład zespołu z rolami
- ☐ specyfikacja: wymagania MoSCoW, wymagania нефункционаłne, 3 przypadki użycia
- ☐ ERD, skrypt baza.sql (działa bez błędów) i 3 zapytania z wynikami
- ☐ makiety 3 ekranów (telefon i komputer), zrzut listy Issues z kamieniami milowymi
- ☐ odpowiedzi na pytania (indywidualnie)

Pytania kontrolne

1. Czym różni się wymaganie funkcjonalne od нефункционального? Podaj przykład z waszego projektu.
2. Jak zapisaliście w bazie relację N–M (jeśli występuje) lub dlaczego jej nie ma?
3. Jaka reguła biznesowa jest w waszej aplikacji i jak ją sprawdzicie w SQL?
4. Dlaczego warto zrobić makiety przed pisanem kodu?

Kryteria oceny

Kryterium	Punkty
Wymagania funkcjonalne (min. 8, MoSCoW) i нефункциональные (min. 4)	3
3 przypadki użycia ze scenariuszem głównym i alternatywnym	3
ERD zgodny ze skryptem SQL; klucze, ograniczenia, indeks	4
Skrypt SQL z danymi testowymi (hasła haszowane) i 3 zapytania z wynikami	3
Makiety 3 ekranów w dwóch wersjach	2
Repozytorium: struktura, .gitignore, Milestones, min. 10 Issues, PR każdej osoby	3
Odpowiedzi na pytania kontrolne (indywidualnie)	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt końcowy cz. 2 — backend

Czas: 2 × 45 min

praca w zespołach 2–3-osobowych (każda osoba na własnej gałęzi funkcji)

Cel: Zbudowanie zaplecza aplikacji zespołowej: logowanie i role, CRUD głównego zasobu, reguła biznesowa, walidacja po stronie serwera i zabezpieczenia wg listy kontrolnej, potwierdzone planem testów ręcznych.

Po zajęciach potrafisz:

- implementuje rejestrację, logowanie i wylogowanie z haszowaniem haseł i odświeżaniem sesji
- ogranicza dostęp do stron i rekordów według roli i właściciela
- programuje CRUD z zapytaniami przygotowanymi, walidacją i komunikatami flash (PRG)
- zabezpiecza formularze tokenem CSRF i escapuje dane wyjściowe
- planuje i wykonuje testy ręczne oraz dokumentuje wyniki

Sprzęt i materiały

- komputer z XAMPP, VS Code i Git;
- repozytorium zespołu z zadania 31
- folder `r2_31_projekt` z Pliki_do_zadan (lista kontrolna bezpieczeństwa, plan testów)

Oprogramowanie

- **XAMPP** — PHP 8.2+, MariaDB, phpMyAdmin
apachefriends.org/download.html
- **Visual Studio Code** — rozszerzenie PHP Intelephense (opcjonalnie)
code.visualstudio.com
- **Dokumentacja PHP — PDO i hasła** — password_hash:
https://www.php.net/password_hash
php.net/manual/pl/book.pdo.php

Przygotowanie przed zajęciami

- Zaktualizujcie specyfikację o uwagi nauczyciela z zadania 31; każda osoba wybiera swoje Issues z kamienia „cz. 2 backend”.
- Przypomnij sobie zadania 9 (PDO CRUD), 12 (logowanie) i 14 (lista kontrolna bezpieczeństwa).

Bezpieczeństwo

- Testuj zabezpieczenia wyłącznie na własnej, lokalnej instalacji — sprawdzasz, czy aplikacja poprawnie odrzuca nieprawidłowe żądania, a nie próbujesz łamać cudzych systemów.
- Hasła kont testowych zapisuj tylko w README jako dane demonstracyjne; nie używaj swoich prawdziwych haseł.

Przebieg zadania

1 Wspólna podstawa (lider)

Jeden plik dołączany na początku każdej strony: konfiguracja z .env (jak w zadaniu 29), połączenie PDO, sesja, nagłówki bezpieczeństwa i funkcje pomocnicze.

```
// src/bootstrap.php – wymagane funkcje
e($s) // htmlspecialchars
przekieruj($adres) // header Location + exit
flash($tekst, $typ) // komunikat na jedno wyświetlenie
wymagaj_logowania(), wymagaj_admina()
csrf_token(), csrf_pole(), sprawdz_csrf()
```

1. Napisz `src/bootstrap.php` z funkcjami z ramki. Ciasteczko sesji ustaw przez `session_set_cookie_params` (httponly, samesite Lax, secure przy HTTPS) przed `session_start()`.
2. Wyślij nagłówki `X-Content-Type-Options: nosniff`, `Referrer-Policy` i `Content-Security-Policy` dopuszczający tylko własne pliki i używany CDN. Sprawdź je w F12 → Network.
3. Przygotuj wspólny nagłówek i stopkę strony (`src/widoki/`) z menu zależnym od roli i miejscem na komunikat flash. Scal przez pull request — pozostali zaczynają od tej wersji.

2 Konta i role

1. Rejestracja: walidacja loginu (wzorzec), długości pól, hasła (min. 10 znaków) i powtórzenia hasła; komunikaty przy polach, zachowanie wpisanych wartości (bez hasła), sprawdzenie unikalności loginu.
2. Logowanie z `password_verify` i `session_regenerate_id(true)`; ten sam komunikat przy złym loginie i złym hasle. Wylogowanie formularzem POST z tokenem CSRF.
3. Zabezpiecz strony: użytkownik niezalogowany → przekierowanie do logowania, zwykły użytkownik na stronie administratora → kod 403. Sprawdzenie ma być po stronie serwera na każdej stronie panelu.

3 Główny zasób i reguła biznesowa

Przykład z projektu wzorcowego: rezerwacja sprzętu odrzucana, gdy koliduje z inną rezerwacją (oczekuje lub zatwierdzona), zaczyna się w przeszłości albo trwa ponad 14 dni.

1. Administrator: lista, dodawanie, edycja i usuwanie głównego zasobu (jedna strona z formularzem lub osobne). Walidacja wszystkich pól na serwerze; błąd unikalności (kod 23000) zamień na komunikat przy polu.
2. Usuwanie i zmiany stanu wyłącznie metodą POST z tokenem CSRF i potwierdzeniem; po zapisie przekierowanie z komunikatem flash (PRG).
3. Użytkownik: tworzenie własnych rekordów (np. rezerwacji) z regułą biznesową sprawdzaną w osobnej funkcji, lista „moje ...” i anulowanie tylko własnych rekordów — warunek `uzytkownik_id = ?` z id z sesji, nie z formularza.
4. Administrator: zmiana statusu rekordów (np. zatwierdź / odrzuć / zwrot) z dozwolonymi przejściami (UPDATE ... WHERE status = poprzedni).

4 Lista kontrolna i testy

1. Przejdź listę `lista_kontrolna_bezpieczenstwa.txt` dla całego kodu zespołu: przy każdym punkcie wpisz plik i linię, która go spełnia, albo utwórz Issue z poprawką.
2. Uzupełnij plan `testy_reczne.txt` (min. 12 przypadków, w tym T9–T11: reguła biznesowa, formularz bez tokenu, zmiana cudzego rekordu) i wykonaj testy. Błędy zgłoś jako Issues i popraw.
3. Każda osoba scala swoje funkcje przez pull request z przeglądem drugiej osoby. Na koniec oznacz wersję tagiem `v0.2-backend`.

W sprawozdaniu umieść

- ☐ lista funkcji backendu z podziałem na osoby i odnośnikami do pull requestów
- ☐ wypełniona lista kontrolna bezpieczeństwa (plik i linia przy każdym punkcie)
- ☐ plan testów ręcznych z wynikami i zrzutami T2, T4, T9, T10
- ☐ fragment kodu reguły biznesowej i sprawdzania właściciela rekordu
- ☐ odpowiedzi na pytania (indywidualnie)

Pytania kontrolne

1. Dlaczego ukrycie przycisku „Panel” w menu nie wystarcza do ochrony strony administratora?
2. Po co `session_regenerate_id(true)` zaraz po zalogowaniu?
3. Dlaczego usuwanie rekordu nie powinno działać przez zwykły odnośnik GET?
4. Dlaczego id użytkownika przy zapisie bierze się z sesji, a nie z ukrytego pola formularza?

Kryteria oceny

Kryterium	Punkty
bootstrap.php: PDO z .env, sesja z bezpiecznymi parametrami, nagłówki, funkcje pomocnicze; wspólny układ stron	3
Rejestracja z walidacją, logowanie z <code>password_verify</code> i regeneracją sesji, wylogowanie POST	3
Kontrola dostępu: przekierowanie niezalogowanych, 403 dla roli, ochrona cudzych rekordów	3
CRUD głównego zasobu z walidacją, PRG, flash, obsługą błędów unikalności	4
Reguła biznesowa i zmiana statusów z dozwolonymi przejściami	2
Lista kontrolna bezpieczeństwa i min. 12 testów ręcznych z wynikami; PR każdej osoby, tag	3
Odpowiedzi na pytania kontrolne (indywidualnie)	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt końcowy cz. 3 — frontend i integracja

Czas: 2 × 45 min

praca w zespołach 2–3-osobowych

Cel: Zbudowanie responsywnego i dostępnego interfejsu aplikacji zespołowej (Bootstrap lub React), połączenie go z backendem przez endpoint JSON i fetch oraz poprawa kodu po przeglądzie.

Po zajęciach potrafisz:

- buduje responsywny interfejs z komponentów Bootstrap (lub React) z aktywną pozycją menu
- projektuje endpoint JSON z walidacją parametrów i właściwymi kodami odpowiedzi
- pobiera dane przez fetch z obsługą ładowania, błędu i przerywania żądania oraz bezpiecznie wstawia je do DOM
- stosuje stopniowe ulepszanie: strona działa bez JS, a skrypt dodaje wygodę
- przeprowadza przegląd kodu wg listy kontrolnej i wprowadza poprawki

Sprzęt i materiały

- komputer z XAMPP, VS Code, Git, Node.js LTS (przy wariancie React); repozytorium z zadań 31–32
- folder `r2_31_projekt` z Pliki_do_zadan (lista przeglądu kodu)

Oprogramowanie

- **Bootstrap 5.3** — wariant podstawowy; wersja React — Vite jak w zadaniach 22–23 getbootstrap.com/docs/5.3/getting-started/introduction
- **Narzędzia F12 (Chrome/Edge)** — Network (fetch), tryb urządzeń, Lighthouse developer.chrome.com/docs/devtools
- **Thunder Client / curl** — test endpointu JSON (jak w zadaniu 18) thunderclient.com

Przygotowanie przed zajęciami

- Scalcie wszystkie gałęzie backendu (tag v0.2-backend) i uzupełnijcie Issues kamienia „cz. 3 frontend”.
- Przypomnij sobie z zadań 4 i 18 fetch z obsługą błędów, a z zadania 25 komponenty Bootstrapa.

Bezpieczeństwo

- Dane z API wstawiaj przez `textContent` lub `createElement` — nigdy przez `innerHTML` z danymi od użytkowników.
- Pliki z CDN dołączaj z atrybutem `integrity` i dopisz domenę CDN do nagłówka `Content-Security-Policy` zamiast go wyłączać.

Przebieg zadania

1 Układ i komponenty

Wariant React: ten sam zakres, komponenty zamiast szablonów PHP; endpointy JSON z kroku 2 są wtedy jedynym źródłem danych.

1. Dołącz Bootstrap (CDN z integrity albo npm) we wspólnym nagłówku. Zbuduj navbar zwijany na telefonie z pozycjami zależnymi od roli; bieżąca pozycja ma klasę `active` i `aria-current="page"`.
2. Przebuduj listę głównego zasobu na siatkę kart (1/2/3 kolumny), formularze na `form-control` / `form-select` z etykietami i komunikatami walidacji przy polach (`is-invalid`, `invalid-feedback`), tabele panelu na `table` w `table-responsive`.
3. Komunikaty flash wyświetl jako `alert` z `role="status"`; dodaj odnośnik „Przejdź do treści” (`visually-hidden-focusable`) i własny kolor marki przez zmienne `--bs-btn-*`. Sprawdź nagłówek CSP — Bootstrap z CDN ma się ładować bez błędów w konsoli.

2 Endpoint JSON

Przykład: `GET api/sprzet.php?q=&kat=&dzien=2026-11-04` zwraca listę sprzętu z polem `wolny` dla wybranego dnia.

```
{
  "dzien": "2026-11-04",
  "liczba": 3,
  "sprzet": [ { "id": 1, "nazwa": "...", "kategoria": "...", "stan": "sprawny", "wolny": false },
  ... ]
}
```

1. Napisz endpoint zwracający dane potrzebne interfejsowi: nagłówek `Content-Type: application/json; charset=utf-8`, `json_encode` z `JSON_UNESCAPED_UNICODE`, typy liczbowe i logiczne (nie same teksty).
2. Waliduj parametry (np. format daty, długość frazy) — błąd zwraca 422 z polem `bład`; inna metoda niż GET → 405 z nagłówkiem Allow. Endpoint korzysta z tych samych zapytań przygotowanych i reguły biznesowej co backend.
3. Przetestuj endpoint w Thunder Client lub curl: poprawne zapytanie, zły parametr, zła metoda. Zapisz kody i fragmenty odpowiedzi.

3 Interakcje JS i fetch

Stopniowe ulepszanie: formularz filtrów działa także bez JS (GET), a skrypt przejmuje go i odświeża tylko listę.

1. W pliku JS przechwyc `submit` i `change` formularza filtrów, pobierz dane z endpointu (`URLSearchParams` z `FormData`), pokaż stan „Ładowanie...” w elemencie `aria-live="polite"`, a przy błędzie komunikat z treścią z pola `bład`.
2. Zbuduj karty funkcją tworzącą elementy (`createElement`, `textContent`); dodaj plakietki stanu (np. wolny/zajęty w wybranym dniu) i licznik wyników. Wyszukiwanie wywołuj z opóźnieniem 300 ms po wpisaniu (debounce), poprzednie żądanie przerywaj `AbortController`.
3. Aktualizuj adres strony `history.replaceState`, aby filtry można było skopiować w linku. Sprawdź w F12 → Network, że przy pisaniu nie powstają dziesiątki żądań.
4. Dodaj drobne interakcje: potwierdzenie przed usunięciem (formularze z klasą `potwierdz`) i kontrolę dat w formularzu (`setCustomValidity`, data „do” nie wcześniejsza niż „od”). Wyłącz JS w przeglądarce i sprawdź, że strona nadal działa.

4 Responsywność, dostępność i przegląd kodu

1. Sprawdź wszystkie ekrany przy 360, 768 i 1280 px oraz obsługę klawiaturą (Tab, Enter, Esc w menu). Uruchom Lighthouse Accessibility dla 2 stron i popraw uwagi (cel ≥ 90).
2. Każda osoba otwiera pull request ze swoją częścią; recenzent przechodzi `lista_przeglądu_kodu.txt` i zostawia min. 3 komentarze do linii. Autor poprawia i odpowiada na komentarze, potem scalenie.
3. Oznacz wersję tagiem `v0.3-frontend` i zrób zrzuty głównych ekranów (telefon i komputer) do folderu `docs/`.

W sprawozdaniu umieść

- ☐ zrzuty głównych ekranów przy 360 i 1280 px, w tym rozwinięte menu
- ☐ opis endpointu (metoda, adres, parametry, przykładowa odpowiedź, kody błędów) i wyniki testu w Thunder Client/curl
- ☐ fragment kodu fetch z obsługą błędów i budowaniem elementów
- ☐ wynik Lighthouse Accessibility dla 2 stron i lista poprawek
- ☐ odnośniki do PR z komentarzami przeglądu i wprowadzonymi poprawkami, odpowiedzi na pytania

Pytania kontrolne

1. Na czym polega stopniowe ulepszanie i co zyskuje użytkownik, gdy JS się nie ładuje?
2. Dlaczego dane z serwera wstawiasz przez `textContent`, a nie `innerHTML`?
3. Po co `debounce` i `AbortController` przy wyszukiwaniu na żywo?
4. Dlaczego endpoint zwraca 422, a nie 200 z pustą listą, gdy data ma zły format?

Kryteria oceny

Kryterium	Punkty
Responsywny układ Bootstrap/React: navbar z aktywną pozycją, karty, formularze z walidacją, tabele, alerty	4
Endpoint JSON: nagłówki, typy danych, walidacja (422), metoda (405), test	4
fetch: ładowanie, błąd, AbortController, debounce, bezpieczne budowanie DOM, <code>replaceState</code>	4
Działanie bez JS, potwierdzenie usuwania, kontrola dat	2
Dostępność (Lighthouse ≥ 90 , klawiatura) i przegląd kodu z poprawkami w PR	4
Odpowiedzi na pytania kontrolne (indywidualnie)	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).

Projekt końcowy cz. 4 — wdrożenie, dokumentacja i prezentacja

Czas: 2 × 45 min

praca w zespołach 2–3-osobowych, prezentacje przed klasą

Cel: Wdrożenie aplikacji zespołowej na serwer, przygotowanie dokumentacji (README, instrukcja użytkownika, opis API i bazy), audyt Lighthouse oraz prezentacja projektu z demonstracją na żywo.

Po zajęciach potrafisz:

- wdraża aplikację PHP + MySQL zgodnie z listą kontrolną i wykonuje test po wdrożeniu
- pisze README, instrukcję użytkownika i opis API dla różnych odbiorców
- mierzy jakość aplikacji (Lighthouse) i poprawia najważniejsze uwagi
- przygotowuje i przeprowadza 5–7-minutową prezentację z demonstracją
- ocenia projekt własny i innych zespołów według kryteriów

Sprzęt i materiały

- komputer z XAMPP, Git i FileZillą; repozytorium projektu (tag v0.3-frontend)
- konto na serwerze szkolnym lub hostingu z PHP 8, MySQL/MariaDB, SFTP i HTTPS
- projektor; folder `r2_31_projekt` (szablon_README.md, kryteria_prezentacji.txt)

Oprogramowanie

- **FileZilla Client** — SFTP; alternatywa WinSCP
filezilla-project.org/download.php?type=client
- **Lighthouse w Chrome/Edge** — F12 → Lighthouse; PageSpeed Insights dla adresu publicznego: <https://pagespeed.web.dev/developer.chrome.com/docs/lighthouse/overview>
- **Markdown w VS Code** — podgląd README: Ctrl+Shift+V
code.visualstudio.com/docs/languages/markdown

Przygotowanie przed zajęciami

- Zamknijcie Issues z kamienia „cz. 3 frontend” lub przenieście je do „Could”. Ustalcie scenariusz demo i podział mówienia.
- Odbierz dane dostępowe do serwera (SFTP, baza) — przypomnij sobie zadanie 29.

Bezpieczeństwo

- Przed wdrożeniem zmień lub usuń konta demonstracyjne; w dokumentacji publicznej nie podawaj haseł produkcyjnych.
- Plik `.env` z danymi produkcyjnymi tworzysz tylko na serwerze — nie trafia do repozytorium ani na slajdy.
- Podczas demo nie pokazuj panelu hostingu ani pliku `.env` na projektorze.

Przebieg zadania

1 Wdrożenie

Postępuj według listy kontrolnej z zadania 29 (docs/WDROZENIE.md w projekcie wzorcowym jest przykładem).

1. Utwórz bazę i konto aplikacji z minimalnymi uprawnieniami, zaimportuj skrypt SQL. Wyślij pliki przez SFTP (bez .env, logów i .git); katalog główny = public/ lub .htaccess z blokadą katalogów prywatnych.
2. Utwórz na serwerze .env z `APP_ENV=production`, sprawdź HTTPS i przekierowanie z http. Zmień hasła kont demonstracyjnych.
3. Wykonaj test po wdrożeniu: obie role, główny CRUD, reguła biznesowa, endpoint JSON (200 i błąd), dostęp do `/.env` i `/src/` (403/404), polskie znaki. Wyniki zapisz w tabeli; błędy zgłoś jako Issues i popraw.
4. Opcjonalnie: przygotuj Dockerfile i docker-compose.yml (PHP + MariaDB) jako sposób uruchomienia lokalnego i opisz go w README.

2 Audyt jakości

1. Uruchom Lighthouse (Mobile, okno InPrivate) dla strony głównej i jednego formularza na wersji wdrożonej. Zapisz 4 wyniki i LCP/CLS.
2. Popraw min. 2 uwagi (np. brak favicon → błąd 404 w konsoli, rozmiar obrazów, kontrast, etykiety, meta description) i zmierz ponownie. Wyniki przed i po wpisz do README.

3 Dokumentacja

Każdy dokument ma innego odbiorcę: README — programista, instrukcja — użytkownik, API — autor klienta (np. aplikacji mobilnej).

1. Uzupełnij `README.md` wg szablonu: opis, adres wdrożenia, funkcje, technologie, uruchomienie lokalne krok po kroku, konta demo (tylko lokalnie), struktura katalogów, bezpieczeństwo, wyniki audytu, zespół.
2. Napisz `docs/INSTRUKCJA.md` dla każdej roli (kroki głównych czynności i tabela najczęstszych komunikatów) oraz `docs/API.md` (metoda, adres, parametry z typami, przykładowa odpowiedź, kody błędów, przykład curl).
3. Dodaj do `docs/` ERD, specyfikację z zadania 31 (zaktualizowaną: co zrealizowano, co przeniesiono do Could/Won't) i zrzuty ekranu. Oznacz wersję tagiem `v1.0` i utwórz Release z opisem zmian.

4 Prezentacja i ocena

Czas: 5–7 min + 2–3 min pytań. Przebieg i arkusz oceny koleżeńskiej — `kryteria_prezentacji.txt`.

1. Przygotuj scenariusz demo (kto, co klika, jakie dane) i maks. 6 slajdów: problem, architektura (ERD, endpoint), bezpieczeństwo i Lighthouse, podział pracy i wnioski. Przygotuj plan B (nagranie ekranu lub wersja na XAMPP).
2. Przeprowadź prezentację: każda osoba mówi, demo na wersji wdrożonej. Odpowiedz na pytania klasy i nauczyciela.
3. Wypełnij arkusze oceny koleżeńskiej dla 2 innych zespołów, a po zajęciach napisz indywidualną refleksję (co zrobisz, czego się nauczyłeś, co poprawiłbyś).

W sprawozdaniu umieść

- ☐ adres wdrożonej aplikacji, tag v1.0 i Release
- ☐ tabela testu po wdrożeniu i wyniki Lighthouse przed/po
- ☐ README, docs/INSTRUKCJA.md, docs/API.md w repozytorium
- ☐ slajdy lub scenariusz demo, arkusze oceny koleżeńskiej

☐ indywidualna refleksja i odpowiedzi na pytania

Pytania kontrolne

1. Czym różnią się odbiorcy README, instrukcji użytkownika i opisu API? Podaj po jednej informacji, która pasuje tylko do jednego dokumentu.
2. Co sprawdziliście po wdrożeniu, czego nie da się sprawdzić na XAMPP?
3. Która reguła bezpieczeństwa była najtrudniejsza do wdrożenia w waszym projekcie i dlaczego?
4. Co zrobilibyście inaczej, zaczynając projekt od nowa?

Kryteria oceny

Kryterium	Punkty
Wdrożenie: działa przez HTTPS, .env na serwerze, zablokowane pliki prywatne, zmienione konta demo	4
Test po wdrożeniu i audyt Lighthouse przed/po z min. 2 poprawkami	3
README wg szablonu, instrukcja użytkownika, opis API, aktualna specyfikacja; tag v1.0 i Release	5
Prezentacja: czas, demo bez błędów, udział każdej osoby, odpowiedzi na pytania	4
Ocena koleżeńska 2 zespołów i indywidualna refleksja	2
Odpowiedzi na pytania kontrolne (indywidualnie)	2
Razem	20

Sprawozdanie przygotuj na wzorze dołączonym do zadania (plik .docx o tym samym numerze).