Technikum w Zespole Szkół im. Armii Krajowej Obwodu “Głuszec” - Grójec w Grójcu

Wymagania edukacyjne

na poszczególne oceny szkolne z przedmiotu:
Laboratorium aplikacji internetowych

I. **Podstawa prawna**

1. Ustawa z dnia 7 września 1991 r. o systemie oświaty (tekst jednolity: Dz.U. z 2024 r., poz. 750) - Rozdział 3a
2. Ustawa z dnia 14 grudnia 2016 r. Prawo oświatowe (Dz.U.2023 poz.900)
3. Rozporządzenie Ministra Edukacji Narodowej z dnia 22 lutego 2019 r. w sprawie oceniania, klasyfikowana i promowania uczniów i słuchaczy w szkołach publicznych (tekst jedn.: Dz.U. z 2023 r., poz. 2572)
4. Statut Technikum w Zespole Szkół im. Armii Krajowej Obwodu "Głuszec" - Grójec w Grójcu.
5. Program nauczania dla zawodu Technik Informatyk 351203

Efekty kształcenia z podstawy programowej	**Po zrealizowaniu zajęć uczeń potrafi**
Uczeń:	
Algorytmy i zasady programowania strukturalnego	**analizować proste problemy programistyczne,** **opisać reprezentacje algorytmów,** **analizować proste algorytmy,** **określić zasady programowania strukturalnego,** **aktywnie słuchać włączając się do dyskusji podczas szukania sposobu rozwiązania problemu,** **stosować zasady kultury osobistej i ogólnie przyjęte normy zachowania w swoim środowisku;** **zastosować właściwą technikę twórczego myślenia przy rozwiązaniu problemu;** **wykazywać się kreatywnością w rozwiązywaniu problemów,**
Ocena niedostateczna	• Uczeń nie posiada znajomości podstawowych pojęć związanych z algorytmami (np. nie rozumie, co to jest algorytm, zmienna, instrukcja warunkowa). • Uczeń nie posiada zdolności do napisania nawet prostego algorytmu w pseudokodzie lub w jakimkolwiek języku programowania. • Uczeń nie posiada znajomości zasad programowania strukturalnego, takich jak sekwencja, iteracja, warunki.
Ocena dopuszczająca	• Uczeń potrafi wymienić podstawowe pojęcia związane z algorytmami i programowaniem strukturalnym (np. zmienne, instrukcje warunkowe, pętle). • Uczeń potrafi rozpoznać elementy algorytmu (np. warunek, instrukcję iteracyjną, instrukcję przypisania). • Uczeń potrafi opisać podstawowe zasady programowania strukturalnego, takie jak

	sekwencja (wykonywanie instrukcji po kolei), wybór (warunki), iteracja (pętle).
Ocena dostateczna	• Uczeń potrafi wyjaśnić, jak działają podstawowe konstrukcje programowania strukturalnego, takie jak instrukcje warunkowe (if, else), pętle (for, while). • Uczeń potrafi opisać sposób działania prostego algorytmu (np. algorytm obliczania sumy liczb). • Uczeń potrafi napisać prosty algorytm w pseudokodzie lub języku programowania, używając instrukcji sekwencyjnych, warunkowych i pętli (np. obliczenie średniej z liczb, znajdowanie maksimum). • Uczeń potrafi wyjaśnić różnicę między programowaniem strukturalnym a innymi paradygmatami (np. obiektowym).
Ocena dobra	• Uczeń potrafi napisać program, który wykorzystuje bardziej złożone algorytmy, takie jak sortowanie (np. sortowanie bąbelkowe, wyszukiwanie liniowe). • Uczeń potrafi zastosować algorytmy rozwiązujące typowe problemy programistyczne (np. algorytmy rekurencyjne, algorytmy operujące na tablicach). • Uczeń potrafi tworzyć i testować programy, stosując zasady programowania strukturalnego (modułowość, brak cyklicznych zależności, prostota kodu). • Uczeń potrafi używać funkcji do podziału problemów na mniejsze części (np. tworzenie funkcji do obliczeń matematycznych lub operacji na danych).
Ocena bardzo bobra	• Uczeń potrafi analizować różne algorytmy pod kątem złożoności czasowej i pamięciowej (np. porównanie złożoności algorytmów sortujących: O(n^2) dla sortowania bąbelkowego, O(n log n) dla quicksorta). • Uczeń potrafi optymalizować kod algorytmów, np. poprzez unikanie zagnieżdżonych pętli, stosowanie odpowiednich struktur danych. • Uczeń potrafi zdiagnozować i poprawić błędy w algorytmach, w tym błędy logiczne (np. nieprawidłowo działające warunki lub nieskończone pętle). • Uczeń potrafi projektować bardziej zaawansowane algorytmy, które rozwiązują skomplikowane problemy, np. algorytmy wyszukiwania (binarny, DFS, BFS).
Ocena celująca	• Uczeń potrafi tworzyć i oceniać zaawansowane algorytmy, takie jak algorytmy sortowania, wyszukiwania, operacji na grafach (np. Dijkstra, Kruskal). • Uczeń potrafi rozwiązywać problemy programistyczne, projektując własne algorytmy na podstawie analizy wymagań problemu.

	• Uczeń potrafi przeprowadzać ewaluację różnych algorytmów pod kątem ich efektywności i stosowności do danego problemu (np. dobór algorytmu w zależności od rodzaju danych i dostępnych zasobów). • Uczeń potrafi tworzyć optymalne rozwiązania problemów programistycznych z wykorzystaniem zaawansowanych konstrukcji programowania strukturalnego (np. rekurencja, dynamiczne programowanie, algorytmy równoległe).
Skryptowe języki programowania	**identyfikować skryptowe języki programowania** **implementować proste algorytmy w języku interpretowanym** **rozróżniać polecenia języka JavaScript i jednego z języków wykonywanych po stronie serwera: PHP, Python, JSP, ASP.NET,** **wyjaśnić pojęcie zmiennej,** **omówić ogólne zasady nazywania zmiennych w JavaScript i w jednym z języków wykonywanych po stronie serwera: PHP, Python, JSP, ASP.NET,** **rozróżniać typy proste zmiennych (liczbowe, tekstowe, logiczne),** **dokonać konwersji typów prostych,** **rozróżniać operatory, ich priorytety i typy (przypisania, porównania, arytmetyczne, logiczne, bitowe, [inkrementacji / dekrementacji]),** **stosować instrukcje warunkowe if...else, [switch],** **realizować powtórzenia w skryptach za pomocą pętli for, while i [do...while],** **tworzyć skrypty w języku JavaScript i w jednym z języków wykonywanych po stronie serwera: PHP, Python, JSP, ASP.NET z wykorzystaniem instrukcji warunkowych i pętli,** **definiować własne funkcje bezargumentowe,** **wyjaśnić pojęcie rekurencji,** **definiować złożone typy danych (tablice, obiekty)**
Ocena niedostateczna	• Uczeń nie posiada wiedzy na temat podstawowych pojęć związanych ze skryptowymi językami programowania. • Uczeń nie posiada umiejętności napisania nawet prostego skryptu w wybranym języku programowania (np. JavaScript, Python). • Uczeń nie posiada znajomości zasad działania skryptów i ich praktycznego zastosowania.
Ocena dopuszczająca	• Uczeń potrafi wymienić popularne skryptowe języki programowania (np. Python, JavaScript, PHP). • Uczeń potrafi rozpoznać podstawowe składniki skryptu (np. zmienne, funkcje, pętle, instrukcje warunkowe). • Uczeń potrafi wymienić typowe zastosowania skryptowych języków

	programowania (np. automatyzacja zadań, obsługa zdarzeń na stronach internetowych).
Ocena dostateczna	• Uczeń potrafi wyjaśnić, czym są skryptowe języki programowania i jak różnią się od języków kompilowanych (np. C++ vs JavaScript). • Uczeń potrafi opisać sposób działania skryptu krok po kroku (np. działanie pętli, instrukcji warunkowych). • Uczeń potrafi napisać prosty skrypt w wybranym języku programowania (np. skrypt, który wyświetla komunikat lub oblicza wartość). • Uczeń potrafi zrozumieć podstawowe konstrukcje językowe, takie jak pętle, funkcje, zmienne, operatory logiczne.
Ocena dobra	• Uczeń potrafi napisać bardziej złożony skrypt, który rozwiązuje konkretny problem (np. kalkulator w JavaScript, parsowanie danych w Pythonie). • Uczeń potrafi zastosować skryptowy język programowania do automatyzacji prostych zadań (np. pobieranie danych z pliku, wysyłanie zapytań HTTP). • Uczeń potrafi wykorzystać narzędzia debuggowania do testowania i poprawiania napisanych skryptów (np. korzystanie z konsoli przeglądarki w JavaScript). • Uczeń potrafi stworzyć skrypt, który działa dynamicznie na stronie internetowej (np. walidacja formularza za pomocą JavaScript).
Ocena bardzo bobra	• Uczeń potrafi analizować bardziej złożone skrypty, zrozumieć ich strukturę i poprawiać błędy logiczne. • Uczeń potrafi rozbić problem programistyczny na mniejsze części i rozwiązać go za pomocą dobrze zaprojektowanych funkcji i modułów w skryptowym języku programowania. • Uczeń potrafi napisać skrypty interaktywne, które integrują się z zewnętrznymi API (np. pobieranie danych za pomocą fetch w JavaScript). • Uczeń potrafi porównywać różne podejścia do rozwiązywania problemów (np. porównanie wydajności różnych algorytmów w Pythonie).
Ocena celująca	• Uczeń potrafi projektować i tworzyć zaawansowane skrypty, które rozwiązują złożone problemy (np. tworzenie aplikacji jednostronicowej za pomocą JavaScript lub automatyzacja złożonych zadań za pomocą Pythona). • Uczeń potrafi optymalizować skrypty pod kątem wydajności i czytelności, stosując dobre praktyki programistyczne (np. minimalizacja złożoności kodu, unikanie powtarzalności). • Uczeń potrafi implementować testy

	automatyczne dla napisanych skryptów, aby zapewnić ich poprawne działanie (np. testy jednostkowe w Pythonie, testowanie DOM w JavaScript). • Uczeń potrafi ewaluować różne skryptowe języki programowania pod kątem ich zalet i wad w konkretnych przypadkach użycia (np. wybór języka do projektu na podstawie jego efektywności w danym kontekście).
Skrypty wykonywane po stronie klienta	**rozróżniać właściwości i funkcje obiektów wbudowanych w JavaScript (np. document, window, navigator, location, style, string, Math, Date, Number)** **scharakteryzować obiektowy model dokumentu (DOM),** **opisać zasady dynamicznej zmiany stylu i zawartości strony,** **opisać zasady dynamicznego tworzenia elementów drzewa dokumentu,** **opisać sposoby rejestracji funkcji obsługujących zdarzenie (inline, przypisanie funkcji [anonimowej], słuchacz zdarzeń),** **manipulować elementami strony za pomocą skryptów JavaScript, np. animacje, efekt rollover, menu rozwijane,** **pobierać i przetwarzać dane z okien dialogowych i kontrolek formularzy,** **wykonywać operacje na ciągach tekstowych,** **wyjaśnić pojęcie frameworku,** **wymienić rodzaje frameworków**
Ocena niedostateczna	• Uczeń nie posiada wiedzy na temat podstawowych pojęć i funkcji związanych ze skryptami po stronie klienta (np. JavaScript). • Uczeń nie potrafi napisać ani zrozumieć prostego skryptu wykonywanego po stronie klienta. • Uczeń nie posiada znajomości zastosowań skryptów po stronie klienta (np. brak umiejętności obsługi interakcji użytkownika na stronie).
Ocena dopuszczająca	• Uczeń potrafi wymienić podstawowe pojęcia związane ze skryptami po stronie klienta (np. zmienne, pętle, funkcje, zdarzenia). • Uczeń potrafi rozpoznać i opisać rolę skryptów po stronie klienta (np. obsługa interakcji na stronie internetowej, walidacja formularzy). • Uczeń potrafi rozpoznać podstawowe elementy skryptu (np. zmienne, funkcje, proste operacje matematyczne w skrypcie).
Ocena dostateczna	• Uczeń potrafi wyjaśnić, w jaki sposób skrypty po stronie klienta wpływają na działanie i interakcję użytkownika ze stroną internetową (np. dynamiczne zmiany treści,

	walidacja danych). • Uczeń potrafi opisać, jak działają podstawowe konstrukcje języka (np. pętle, warunki, zmienne) i jakie mają zastosowanie w skryptach po stronie klienta. • Uczeń potrafi napisać prosty skrypt po stronie klienta, który wykonuje określoną funkcję, np. walidacja danych w formularzu lub wyświetlenie komunikatu w reakcji na zdarzenie (kliknięcie przycisku).
Ocena dobra	• Uczeń potrafi napisać skrypt po stronie klienta, który reaguje na interakcje użytkownika (np. skrypt, który zmienia styl elementów na stronie w odpowiedzi na kliknięcia, czy obsługuje formularze). • Uczeń potrafi wykorzystać API przeglądarki (np. document.querySelector, addEventListener) do manipulacji elementami HTML oraz dynamicznej zmiany treści strony. • Uczeń potrafi stworzyć i przetestować skrypt do walidacji danych wprowadzanych przez użytkownika, np. w formularzu rejestracyjnym. • Uczeń potrafi korzystać z konsoli przeglądarki do debugowania błędów w skryptach (np. znajdowanie błędów składniowych i logicznych).
Ocena bardzo bobra	• Uczeń potrafi analizować działanie złożonych skryptów po stronie klienta, rozumiejąc ich strukturę i sposób działania. • Uczeń potrafi optymalizować skrypty pod kątem wydajności i poprawności działania (np. poprawianie problemów z wydajnością, minimalizacja czasu ładowania i interakcji). • Uczeń potrafi tworzyć bardziej zaawansowane skrypty, które wykonują operacje asynchroniczne (np. pobieranie danych z serwera za pomocą AJAX/Fetch API, obsługa asynchronicznych funkcji w JavaScript). • Uczeń potrafi diagnozować i poprawiać błędy oraz nieprawidłowości w skryptach, analizując wyniki testów i logi z konsoli przeglądarki.
Ocena celująca	• Uczeń potrafi projektować zaawansowane, interaktywne aplikacje internetowe z wykorzystaniem skryptów po stronie klienta, które w pełni integrują się z interfejsem użytkownika (np. aplikacje SPA - Single Page Application).

	• Uczeń potrafi tworzyć skrypty, które dynamicznie aktualizują dane na stronie w oparciu o komunikację z serwerem, wykorzystując techniki asynchroniczne (np. WebSockets, Fetch API, AJAX). • Uczeń potrafi implementować optymalne i responsywne interakcje użytkownika, uwzględniając zasady dostępności (Accessibility) i najlepsze praktyki UX (User Experience). • Uczeń potrafi ewaluować i testować skrypty, stosując zaawansowane narzędzia i techniki testowania (np. automatyczne testy jednostkowe, testy wydajności, debugowanie narzędziami developerskimi przeglądarki).
Skrypty wykonywane po stronie serwera	**tworzyć proste programy w jednym z języków wykonywanych po stronie serwera: Python, PHP, JSP, ASP.NET,** **stosować wbudowane instrukcje i funkcje, np. działające na tekstach,** **pobierać i przesyłać dane z formularza,** **opisać sposób współpracy aplikacji z bazą danych (biblioteki),** **objaśnić mechanizm ciasteczek (cookie)**
Ocena niedostateczna	• Uczeń nie posiada wiedzy na temat podstawowych pojęć związanych ze skryptami wykonywanymi po stronie serwera. • Uczeń nie potrafi zrozumieć, co to są skrypty po stronie serwera ani jakie mają zastosowanie. • Uczeń nie potrafi napisać ani zrozumieć nawet prostego skryptu działającego po stronie serwera (np. skryptu obsługującego żądanie HTTP).
Ocena dopuszczająca	• Uczeń potrafi wymienić podstawowe skryptowe języki programowania wykorzystywane po stronie serwera (np. PHP, Python, Node.js). • Uczeń potrafi rozpoznać i opisać ogólną rolę skryptów po stronie serwera w działaniu aplikacji internetowych (np. obsługa zapytań HTTP, zarządzanie bazami danych). • Uczeń potrafi wymienić przykłady zastosowania skryptów po stronie serwera (np. logowanie użytkownika, generowanie dynamicznych stron).
Ocena dostateczna	• Uczeń potrafi wyjaśnić, czym różnią się skrypty po stronie serwera od skryptów po stronie klienta (np. wykonywanie kodu na

	serwerze a w przeglądarce). • Uczeń potrafi zrozumieć podstawowe operacje wykonywane przez skrypty po stronie serwera, takie jak przetwarzanie formularzy, obsługa żądań GET i POST. • Uczeń potrafi napisać prosty skrypt po stronie serwera, np. skrypt odbierający dane z formularza i generujący dynamiczną odpowiedź (w PHP, Node.js itp.). • Uczeń potrafi opisać, jak działa serwer HTTP i jak skrypty po stronie serwera reagują na żądania HTTP.
Ocena dobra	• Uczeń potrafi napisać skrypt po stronie serwera, który obsługuje bardziej złożone operacje, takie jak logowanie użytkownika, walidacja danych czy interakcja z bazą danych. • Uczeń potrafi zastosować techniki sesji i ciasteczek (cookies) do zarządzania stanem aplikacji internetowej. • Uczeń potrafi obsługiwać różne typy żądań HTTP (GET, POST, PUT, DELETE) i generować dynamiczne treści na podstawie tych żądań. • Uczeń potrafi testować i debugować skrypty po stronie serwera, używając odpowiednich narzędzi (np. logów serwera, narzędzi do testowania API jak Postman).
Ocena bardzo bobra	• Uczeń potrafi analizować złożone skrypty po stronie serwera, rozumiejąc ich strukturę i optymalizując je pod kątem wydajności (np. optymalizacja zapytań do bazy danych, caching). • Uczeń potrafi tworzyć skrypty po stronie serwera, które obsługują operacje CRUD (Create, Read, Update, Delete) z wykorzystaniem bazy danych. • Uczeń potrafi wykorzystywać bardziej zaawansowane techniki, takie jak middleware w Node.js czy frameworki MVC (np. Laravel, Express.js), aby strukturyzować kod. • Uczeń potrafi analizować działanie aplikacji serwerowej pod kątem bezpieczeństwa, np. zabezpieczenie skryptów przed atakami typu SQL Injection czy XSS (Cross-Site Scripting).
Ocena celująca	• Uczeń potrafi projektować i implementować zaawansowane aplikacje serwerowe, które komunikują się z bazami danych i API, zapewniając pełną funkcjonalność aplikacji internetowych. • Uczeń potrafi tworzyć bezpieczne i wydajne

	systemy oparte na skryptach po stronie serwera, stosując najlepsze praktyki związane z zabezpieczeniem aplikacji (np. korzystanie z technik szyfrowania danych, zabezpieczenia przed atakami DDoS). • Uczeń potrafi tworzyć aplikacje działające w architekturze mikroserwisów, które komunikują się za pomocą API (np. REST lub GraphQL). • Uczeń potrafi ewaluować różne technologie i narzędzia po stronie serwera pod kątem ich efektywności w różnych przypadkach użycia (np. wybór pomiędzy Node.js, PHP czy Pythonem w zależności od wymagań projektu).
Środowisko programistyczne i uruchomieniowe aplikacji internetowych	**opisać funkcje środowiska programistycznego, tworzyć programy w wybranym środowisku programistycznym scharakteryzować gotowe pakiety dla aplikacji internetowych, np. phpMyAdmin**
Ocena niedostateczna	• Uczeń nie posiada wiedzy na temat środowisk programistycznych oraz uruchomieniowych dla aplikacji internetowych. • Uczeń nie potrafi opisać podstawowych narzędzi wykorzystywanych w tworzeniu i uruchamianiu aplikacji internetowych (np. edytorów kodu, serwerów). • Uczeń nie rozumie roli środowisk uruchomieniowych (np. serwerów HTTP, przeglądarek, systemów operacyjnych).
Ocena dopuszczająca	• Uczeń potrafi wymienić podstawowe narzędzia wykorzystywane w środowisku programistycznym (np. edytory kodu, takie jak Visual Studio Code, Sublime Text) oraz środowiskach uruchomieniowych (np. serwery Apache, Nginx). • Uczeń potrafi rozpoznać podstawowe pojęcia związane ze środowiskami uruchomieniowymi aplikacji internetowych (np. co to jest serwer HTTP, lokalne środowisko developerskie). • Uczeń potrafi opisać ogólną rolę środowiska uruchomieniowego w działaniu aplikacji internetowych (np. hosting aplikacji na serwerze, przetwarzanie kodu HTML, CSS, JavaScript).
Ocena dostateczna	• Uczeń potrafi wyjaśnić różnicę między lokalnym środowiskiem programistycznym a środowiskiem produkcyjnym (np. testowanie na lokalnym serwerze a

	publikacja na serwerze produkcyjnym). • Uczeń potrafi zrozumieć rolę serwera w uruchamianiu aplikacji internetowej oraz jak serwer komunikuje się z przeglądarką. • Uczeń potrafi zainstalować i skonfigurować lokalne środowisko uruchomieniowe (np. XAMPP, WAMP, MAMP) oraz uruchomić prostą aplikację internetową. • Uczeń potrafi opisać, jak działa proces wdrażania aplikacji internetowej na serwerze (np. przesyłanie plików przez FTP, uruchamianie aplikacji na hostingu).
Ocena dobra	• Uczeń potrafi samodzielnie skonfigurować pełne środowisko programistyczne do tworzenia aplikacji internetowych, korzystając z narzędzi takich jak IDE, serwery lokalne, bazy danych oraz systemy kontroli wersji (np. Git). • Uczeń potrafi uruchomić aplikację internetową na serwerze lokalnym i przetestować jej działanie. • Uczeń potrafi zastosować narzędzia do debugowania aplikacji internetowych, takie jak konsola przeglądarki, narzędzia developerskie, oraz lokalne logi serwera. • Uczeń potrafi wdrażać aplikacje internetowe na zdalnym serwerze, korzystając z narzędzi takich jak FTP, SSH, lub systemów CI/CD (Continuous Integration/Continuous Deployment).
Ocena bardzo bobra	• Uczeń potrafi analizować i optymalizować środowisko programistyczne i uruchomieniowe, dostosowując je do specyficznych wymagań projektu (np. wybór serwera, konfiguracja środowiska developerskiego i produkcyjnego, zarządzanie zależnościami). • Uczeń potrafi tworzyć zaawansowane konfiguracje środowisk uruchomieniowych, w tym stosowanie kontenerów (np. Docker) do izolowania aplikacji i zarządzania ich wersjami. • Uczeń potrafi optymalizować środowisko produkcyjne pod kątem wydajności aplikacji internetowych (np. caching, load balancing, konfiguracja baz danych). • Uczeń potrafi zastosować techniki monitorowania i logowania aplikacji w środowisku produkcyjnym, aby śledzić wydajność aplikacji i szybko diagnozować problemy.

Ocena celująca	• Uczeń potrafi projektować i implementować pełne środowiska programistyczne i uruchomieniowe aplikacji internetowych, stosując najlepsze praktyki dotyczące skalowalności, wydajności i bezpieczeństwa. • Uczeń potrafi tworzyć systemy CI/CD, które automatyzują wdrażanie aplikacji internetowych, testowanie oraz monitorowanie ich stanu w środowisku produkcyjnym. • Uczeń potrafi projektować zaawansowane rozwiązania uruchomieniowe z wykorzystaniem chmury obliczeniowej (np. AWS, Google Cloud, Azure), korzystając z usług takich jak serwery wirtualne, bazy danych w chmurze oraz systemy do zarządzania ruchem sieciowym (np. CloudFront). • Uczeń potrafi ewaluować i wybierać najbardziej efektywne technologie środowisk uruchomieniowych dla konkretnych aplikacji (np. różnica między serwerami tradycyjnymi a rozwiązaniami typu serverless).
Walidacja kodu programu	**opisać sposoby testowania tworzonych programów, poprawiać błędy w tworzonych programach,**
Ocena niedostateczna	• Uczeń nie posiada wiedzy na temat walidacji kodu programu. • Uczeń nie potrafi wymienić ani zrozumieć, czym jest walidacja kodu i jakie są jej cele. • Uczeń nie potrafi przeprowadzić nawet najprostszej walidacji kodu ani zidentyfikować błędów.
Ocena dopuszczająca	• Uczeń potrafi zdefiniować, czym jest walidacja kodu programu. • Uczeń potrafi wymienić podstawowe techniki walidacji kodu (np. sprawdzanie poprawności składni, testy jednostkowe). • Uczeń potrafi rozpoznać narzędzia używane do walidacji kodu (np. lintery, narzędzia do testów automatycznych, walidatory HTML/CSS/JavaScript).
Ocena dostateczna	• Uczeń potrafi wyjaśnić, dlaczego walidacja kodu jest ważna w procesie tworzenia oprogramowania (np. poprawność, zgodność z normami, eliminacja błędów). • Uczeń potrafi zrozumieć różnicę między walidacją kodu a testowaniem oprogramowania. • Uczeń potrafi użyć narzędzi do podstawowej walidacji kodu (np. HTML/CSS validator,

	ESLint, Pylint) w celu znalezienia błędów syntaktycznych i logicznych. • Uczeń potrafi interpretować wyniki prostych walidacji i naprawiać podstawowe błędy w kodzie.
Ocena dobra	• Uczeń potrafi zastosować techniki walidacji w praktyce, np. uruchamianie testów jednostkowych oraz automatycznych walidatorów kodu w ramach procesu tworzenia aplikacji. • Uczeń potrafi naprawiać błędy znalezione w procesie walidacji kodu (np. naprawa błędów składniowych, eliminacja problemów związanych z dostępnością i kompatybilnością przeglądarek). • Uczeń potrafi zintegrować narzędzia do walidacji z procesem CI/CD (Continuous Integration/Continuous Deployment) w celu automatycznej walidacji kodu. • Uczeń potrafi walidować kod pod kątem zgodności ze standardami branżowymi (np. WCAG, W3C) oraz z ogólnie przyjętymi zasadami dobrych praktyk programowania.
Ocena bardzo bobra	• Uczeń potrafi analizować wyniki walidacji i rozpoznawać bardziej złożone problemy, takie jak nieefektywność kodu, błędy w logice biznesowej, potencjalne problemy z bezpieczeństwem. • Uczeń potrafi zastosować zaawansowane narzędzia do walidacji, takie jak statyczna analiza kodu (np. SonarQube) oraz narzędzia do analizy kodu pod kątem wydajności i bezpieczeństwa. • Uczeń potrafi przeprowadzać ręczną walidację kodu w połączeniu z automatycznymi narzędziami, aby wychwycić błędy, które nie są wykrywane przez narzędzia automatyczne. • Uczeń potrafi porównywać wyniki różnych narzędzi walidacyjnych i proponować usprawnienia w kodzie w celu optymalizacji, poprawy wydajności lub bezpieczeństwa.
Ocena celująca	• Uczeń potrafi projektować i implementować kompleksowe procedury walidacji kodu, integrując narzędzia do walidacji w złożonych projektach (np. używanie zestawów testów jednostkowych, integracyjnych i end-to-end w procesie rozwoju oprogramowania). • Uczeń potrafi tworzyć niestandardowe narzędzia lub skrypty do walidacji

	specyficznych wymagań projektu (np. walidacja specyficznych standardów, które nie są objęte standardowymi narzędziami). • Uczeń potrafi wykorzystywać techniki monitorowania i walidacji kodu w produkcyjnym środowisku aplikacji, w celu wykrywania i naprawiania błędów w czasie rzeczywistym. • Uczeń potrafi ewaluować skuteczność różnych technik walidacji i dostosowywać procesy walidacyjne w celu spełnienia wymagań związanych z jakością, wydajnością i bezpieczeństwem aplikacji.
Dokumentowanie aplikacji	**opisuje sposoby umieszczania komentarzy w kodzie źródłowym programu,**
Ocena niedostateczna	• Uczeń nie posiada wiedzy na temat znaczenia i zasad dokumentowania aplikacji. • Uczeń nie potrafi wymienić podstawowych elementów dokumentacji ani zrozumieć, dlaczego dokumentowanie jest istotne. • Uczeń nie podejmuje żadnych prób dokumentowania aplikacji, lub dokumentacja jest niekompletna i chaotyczna.
Ocena dopuszczająca	• Uczeń potrafi wymienić podstawowe elementy dokumentacji aplikacji (np. instrukcja instalacji, wymagania systemowe, opis funkcji). • Uczeń potrafi rozpoznać znaczenie dokumentacji w procesie tworzenia oprogramowania. • Uczeń potrafi podać przykłady narzędzi i formatów używanych do dokumentowania aplikacji (np. pliki README, formaty markdown, wiki). • Uczeń potrafi wskazać elementy dokumentacji, które są standardowo wymagane w projektach programistycznych.
Ocena dostateczna	• Uczeń potrafi wyjaśnić, dlaczego dokumentacja jest istotna dla użytkowników i programistów (np. ułatwia utrzymanie aplikacji, wspiera nowych użytkowników). • Uczeń potrafi stworzyć podstawową dokumentację aplikacji, obejmującą instalację, konfigurację i ogólny opis funkcjonalności. • Uczeń potrafi posługiwać się narzędziami i formatami dokumentacyjnymi (np. Markdown, HTML) do przygotowania prostej dokumentacji. • Uczeń potrafi zrozumieć i używać

	komentarzy w kodzie jako części dokumentacji (np. objaśnienie trudniejszych fragmentów kodu).
Ocena dobra	• Uczeń potrafi przygotować kompletną dokumentację techniczną aplikacji, w tym instrukcje instalacji, konfiguracji, opis funkcjonalności, wymagania systemowe oraz instrukcje użytkowania. • Uczeń potrafi zastosować najlepsze praktyki w dokumentowaniu kodu, w tym pisanie szczegółowych komentarzy, wyjaśniających logikę kluczowych funkcji. • Uczeń potrafi wykorzystywać narzędzia automatyzujące generowanie dokumentacji (np. Javadoc, Sphinx dla Python) oraz tworzyć pliki README z odpowiednimi sekcjami. • Uczeń potrafi dokumentować procesy związane z obsługą aplikacji, takie jak procedury aktualizacji, backupu i wdrażania nowych wersji.
Ocena bardzo bobra	• Uczeń potrafi analizować istniejącą dokumentację i oceniać jej kompletność oraz czytelność zarówno dla programistów, jak i użytkowników końcowych. • Uczeń potrafi poprawiać i optymalizować dokumentację, wprowadzając ulepszenia w strukturze, języku i treści, aby była bardziej zrozumiała i funkcjonalna. • Uczeń potrafi tworzyć dokumentację zgodną z wytycznymi i standardami branżowymi, np. dokumentacja API zgodna z OpenAPI lub Swagger. • Uczeń potrafi tworzyć dokumentację aplikacji w formie interaktywnej (np. API Docs z możliwością testowania zapytań) oraz dokumentować zaawansowane funkcje, jak integracje z zewnętrznymi systemami.
Ocena celująca	• Uczeń potrafi projektować pełne systemy dokumentacyjne dla złożonych aplikacji, które obejmują nie tylko dokumentację użytkownika, ale również architekturę systemu, diagramy UML, schematy baz danych i dokumentację API. • Uczeń potrafi tworzyć dokumentację wielojęzyczną, dostosowaną do potrzeb różnych grup użytkowników i regionów. • Uczeń potrafi ewaluować istniejące systemy dokumentacyjne i proponować niestandardowe rozwiązania, takie jak wiki,

	bazy wiedzy czy platformy interaktywne do współpracy przy dokumentowaniu projektów (np. Confluence). • Uczeń potrafi tworzyć procedury i wytyczne dla zespołów developerskich w zakresie dokumentowania aplikacji, zgodne z najlepszymi praktykami w zarządzaniu projektami IT (np. Agile, DevOps).