Skuteczne wyszukiwanie błędów, devtools

Podczas egzaminu INF.03 wymaga się od nas tworzenia poprawnego składniowo kodu JavaScript . I dlatego tak ważne jest jego skuteczne debugowanie, czyli znajdowanie i naprawianie błędów. Każdy, kto programował choćby przez chwilę zdaje sobie sprawę, że błędy są nieuniknione. Co ważne (zwłaszcza podczas egzaminu) to umieć je szybko znaleźć i poprawić! *DevTools*, czyli narzędzia deweloperskie wbudowane w przeglądarkę pomogą nam szybko zdebugować kod, przeanalizować działanie poszczególnych linii i przez to skutecznie pozbyć się ewentualnych literówek czy nawet błędów logicznych.

DevTools – jak otworzyć w różnych przeglądarkach

Aby szybko wywołać narzędzia developerskie w nowoczesnej przeglądarce (*Chrome*, *Edge*, *Firefox*), wystarczy użyć skrótów klawiaturowych:

- **F12** lub **Ctrl + Shift + I**
- **Ctrl + Shift + J** – otwarcie od razu konsoli, celem np. zobaczenia komunikatów wypisywanych przez nas w `console.log()`

DevTools składa się z kilku zakładek, ale skupimy się w tej lekcji na trzech najważniejszych dla debugowania:

- **Console** – wyświetla błędy, ostrzeżenia i komunikaty
- **Sources** – pozwala na podgląd i edycję kodu w czasie rzeczywistym
- **Network** – pokazuje zapytania HTTP i ewentualne błędy po stronie serwera

Zakładka *Console* – sprawdzenie komunikatów

Najprostszy sposób na szybkie znalezienie błędów składniowych w kodzie to zajrzenie do konsoli – otrzymamy tam często informacje o naturze problemów. Rozważmy prosty przykład literówki popełnionej w kodzie:

```
let liczby = [2, 4, 6, 8, 10];
let suma = 0;
for (let i = 0; i < liczby.length; i++) {
    sume += liczby[i]; // literówka!
}
alert(suma);
```

Oczywiście zakładamy, iż nie zdajemy sobie sprawy z istnienia tej literówki 🙂 Po uruchomieniu skryptu, w konsoli zobaczymy:

```
Uncaught ReferenceError: sume is not defined
at skrypt.js:5
```

Dzięki temu dowiadujemy się, że w tej konkretnej, piątej linii istnieje błąd – i łatwo zauważyć, że faktycznie w zapisie zmiennej `sume` zrobiliśmy literówkę podczas jej wpisywania. Wykrycie tej trywialnej pomyłki bez zajrzenia do konsoli może zwyczajnie zająć nam więcej czasu. Informacja w której linii skrypt “poddał się” podczas wykonania kodu bywa niezwykle cenna.

Wypisanie wartości zmiennych – console.log()

Najczęściej używana sztuczka podczas sprawdzania działania skryptów to podręczne wypisywanie wartości używanych zmiennych w konsoli. Rozważmy na przykład śledzenie wyznaczanej w pętli sumy liczb:

```
let liczby = [3, 6, 9, 12];
let suma = 0;
for (let i = 0; i < liczby.length; i++) {
    suma += liczby[i];
    console.log("Po dodaniu ", liczby[i], "suma wynosi:", suma);
}
```

Efekt implementacji takiego “raportowania” w konsoli przeglądarki:

```
Po dodaniu 3 suma wynosi: 3
Po dodaniu 6 suma wynosi: 9
Po dodaniu 9 suma wynosi: 18
Po dodaniu 12 suma wynosi: 30
```

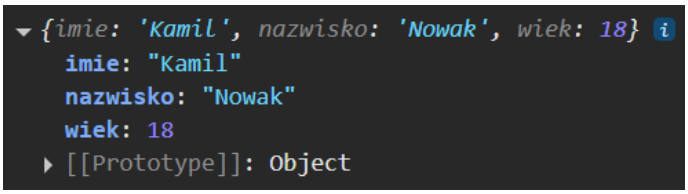
Bardzo szybko możemy wówczas zauważyć, że jakaś liczba np. nie została pobrana z tablicy albo obliczona suma nie rośnie tak jak powinna. Takie “wyrzucanie na ekran” wartości zmiennych to fantastyczny sposób upewnienia się podczas egzaminu, czy nasz skrypt na pewno działa prawidłowo.

Wypisanie w konsoli zawartości obiektu

W konsoli dostępnej w *DevTools* możemy też bardzo wygodnie zobaczyć podgląd obiektu:

```
// Przykładowy obiekt w JS
let uczen = {
    imie: "Kamil",
    nazwisko: "Nowak",
    wiek: 18
};
console.log(uczen); // wypisanie obiektu w czytelnej formie
```

Podgląd obiektu w konsoli:



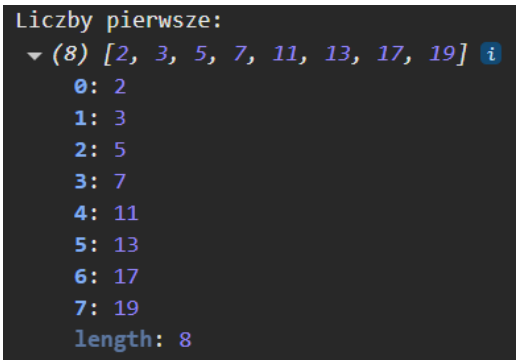
Wypisanie w konsoli zawartości tablicy

Równie wygodnie możemy podejrzeć wartości (szuflady) zgromadzone w tablicy – na przykład gdy generujemy tablicę liczb pierwszych:

```
let liczby = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15, 17, 18, 19, 20];
let pierwsze = [];
function czyPierwsza(n) {
    if (n < 2) return false;
    for (let i = 2; i <= Math.sqrt(n); i++) {
        if (n % i === 0) return false;
    }
    return true;
}
```

```
// przeglądanie tablicy i "zbieranie" liczb pierwszych
for (let i = 0; i < liczby.length; i++) {
    if (czyPierwsza(liczby[i])) {
        pierwsze.push(liczby[i]);
    }
}
// wypisanie całej tablicy z liczbami pierwszymi - po prostu używamy jej nazwy
console.log("Liczby pierwsze:", pierwsze);
```

Podgląd tablicy w konsoli:



Użyteczne, gdy nie jesteśmy pewni, czy tablica jest dobrze wypełniona wartościami generowanymi w skrypcie albo czy iterujemy po właściwych indeksach podczas realizowania jakiegoś algorytmu używającego tablicy.

Obsługa błędów w kodzie: try .. catch

Jeśli chcemy uniknąć takiego wyświetlania błędów w konsoli przeglądarki podczas tworzenia skryptów, to możemy też używać bloku

`try..catch`:

```
let imiona = ["Ania", "Bartek", "Celina", "Damian"];
try {
    // jest tu literówka: zamiast imiona.length wpisano imona.length
    for (let i = 0; i < imona.length; i++) {
        alert("Imię nr " + (i + 1) + ": " + imiona[i]);
    }
} catch (blad) {
    // złapanie wyjątku pozwoli wykryć błąd
    console.log("Wystąpił błąd:", blad.message);
}
```

Zasadniczo, to takie podejście na egzaminie zawodowym kosztuje nas nieco więcej czasu aniżeli proste “console logowanie”, lecz oczywiście jest to fajny sposób sprawdzenia który fragment skryptu nie działa poprawnie w naszym źródle. Zwłaszcza, iż sekcja

`catch` przechwytuje ewentualny błąd, zamiast zatrzymać dalsze wykonanie całego skryptu – to jest główna przewaga tego zapisu.

Debugowanie eksperckie – zakładki *Sources*, *Network*

Teraz przedstawimy kolejne dwie zakładki w *DevTools*, które okażą się mniej użyteczne na egzaminie INF.03, lecz warto wiedzieć o takiej możliwości debugowania kodów w kontekście tworzenia rzeczywistych witryn.

Po pierwsze – zakładka *Sources*. Jeśli nasz kod zawiera funkcje realizujące bardziej złożone algorytmy, to warto zatrzymać jego wykonanie i przeanalizować działanie skryptu krok po kroku:

- Przechodzimy w *DevTools* do zakładki *Sources.*
- Otwieramy wybrany skrypt JavaScript.
- Klikamy w numer linii – ustanawiamy *breakpoint* (punkt przerwania) w kodzie.
- Odświeżamy stronę – wykonanie kodu zatrzyma się na wybranej linii.

Oczywiście zadania egzaminacyjne (z uwagi na stopień złożoności) raczej nie będą wymagały od nas podjęcia tego typu działań debugujących, lecz warto wiedzieć o takiej możliwości. Więcej informacji o stosowaniu punktów wstrzymania wykonania kodu oraz pracy krokowej znajdziesz tutaj: Wstrzymywanie kodu z punktami przerwania.

Druga bardziej zaawansowana technika debugowania to wykorzystanie zakładki *Network*, ponieważ czasami błąd nie jest umiejscowiony w kodzie JavaScript, ale w komunikacji z serwerem. Ponownie – nieprzydatne na egzaminie, lecz warto dowiedzieć się więcej jeśli tworzymy realne witryny: Sprawdzanie aktywności w sieci.

Podsumowanie

Na egzaminie INF.03 możemy mieć zadanie, w którym umiejętności prostego debugowania gotowego kodu pozwolą nam szybko poprawić błędy – warto umieć korzystać z *DevTools* i szybko zdiagnozować problem. Kilka wniosków z lekcji:

- Na egzaminie INF.03 musimy pisać poprawny składniowo kod JavaScript – warto więc znać techniki skutecznego debugowania błędów.
- DevTools (narzędzia developerskie) uruchamiamy skrótem: `F12` lub `Ctrl + Shift + I`, a od razu konsolę: `Ctrl + Shift + J`.
- Zakładka *Console* pokazuje błędy składniowe, typy błędów i wskazuje numery linii, w których wystąpił problem.
- Typowy błąd literówki np. `sume += liczby[i]` zamiast `suma += liczby[i]` szybko znajdziemy gdy zajrzymy do komunikatu: *ReferenceError: sume is not defined*.
- Do sprawdzania wartości zmiennych używajmy prostego `console.log()` – możemy wypisywać wartości i sprawdzać ich poprawność.
- Możemy wypisywać całe obiekty i tablice do konsoli w czytelnej formie.
- Funkcja `try...catch` pozwala wyłapać błędy np. literówki w nazwie zmiennej i nie zatrzymać całego skryptu.
- Przykład: błędny indeks tablicy w pętli można zabezpieczyć blokiem `try...catch` i wypisać komunikat błędu przez proste `blad.message`.
- Zakładka *Sources* umożliwia ustawienie breakpointów i analizę działania kodu krok po kroku – niekonieczne na egzaminie, ale przydatne w praktyce webowej.
- Zakładka *Network* służy do diagnozowania błędów w komunikacji z serwerem – nieistotna na egzaminie, ale ważna przy realnych projektach.