Lekcja 1: Czerwone faliste linie

Spójrz na ten fragment kodu.

```
const storage = {
max: undefined,
items: []
}
Object.defineProperty(storage, 'max', { readonly:
true, val: 5000 })
let currentStorage = 'undefined'
function storageUsed() {
if(currentStorage) {
return currentStorage
}
currentStorage = 0
for(const i = 0; i < storage.length(); i++) {
currentStorage += storage.items[i].weigth
}
}
function add(item) {
if(storage.max - item.weight >= storageUsed) {
storage.items.add(item)
currentStorage += iten.weight
}
}
```

Około 25 linijek JavaScript i dużo się tam dzieje! Jeśli zbadamy ten fragment, możemy wywnioskować, że dotyczy to jakiegoś magazynu, może statku, który może pomieścić około 5000 ton przedmiotów. Pomagają nam w tym dwie funkcje, dodając elementy i sprawdzając, czy zostało jeszcze miejsce. Wygląda na całkiem proste, prawda? Cóż, jest pewien haczyk: podstępny mały błąd ukryty gdzieś w tych 25 wierszach kodu. Czy potrafisz to dostrzec? W porządku, to było małe kłamstwo. Jest nie tylko jeden błąd, jest ich wiele. Jest tak wiele błędów i błędów, że żadna linia nie jest pozbawiona wad. Ten kawałek jest całkowicie zepsuty. I jest gorzej. Nie otrzymujemy żadnych informacji zwrotnych z przeglądarki (ani Node.js) informujących nas, że coś jest nie tak. Ten fragment kodu nie przerywa ani

nie wypełnia konsoli przeglądarki komunikatami o błędach. Pozostaje cicho, gdy wywołujemy metodę add:

add ({weight: 3000})

Brak widocznych błędów. Po prostu nie działa. Witamy w JavaScript! Wiadomo, że takie rzeczy doprowadzają programistów do szaleństwa. Ludzie wątpią w swoje wybory zawodowe, spędzają godziny na dowiadywaniu się, gdzie ich palec wślizgnął się do sąsiedniego klawisza, wykonując zwykle solidną pracę błędną. Takie błędy powodują również złą reputację JavaScript, co jest smutne w przypadku języka programowania, który umożliwia tak wielu z nas tworzenie wspaniałych rzeczy. Im dłużej wpatrujesz się w 25 linijek JavaScript, tym więcej napotkasz problemów. Rzeczy, które są po prostu zepsute, ale także rzeczy, które wydają się tylko nieznacznie nieprawidłowe i zachęcają do przetestowania. Problem polega na tym, że po prostu patrząc na tekst, trudno jest śledzić wszystko, co może być niezwykłe. Czy nie byłoby miło, gdybyśmy mogli uzyskać wizualną informację zwrotną, która pomogłaby nam natychmiast zidentyfikować problemy, zamiast spędzać godziny na ich poszukiwaniu? Pomyśl o wrod - przepraszam - edytorze tekstu wyświetlającym literówki z czerwonymi falistymi liniami.

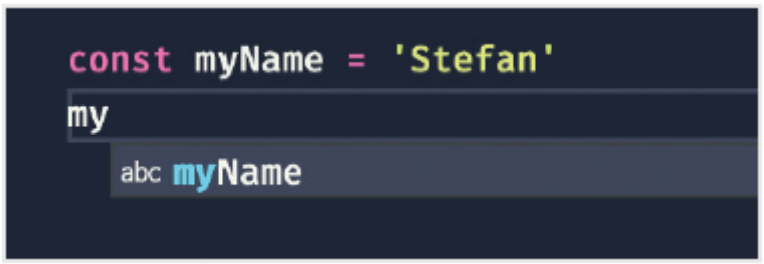
Wprowadź TypeScript

Taki jest cel języka TypeScript: wskazywanie potencjalnych błędów, zanim kod trafi do środowiska

produkcyjnego. Przede wszystkim TypeScript zapewnia analizę kodu JavaScript. Weź nowoczesny edytor kodu. Być może korzystasz z Visual Studio Code (VS Code). Po otwarciu pliku JavaScript program VS Code może nie różnić się zbytnio od innych edytorów. Ale pod spodem TypeScript jest już aktywny. Bieganie, sprawdzanie, analizowanie. Potrzebujesz dowodu? Utwórz nowy plik JavaScript i wprowadź następujący wiersz:

var myName = 'Stefan'

Oczywiście możesz używać swojego imienia i nazwiska. Teraz wpisz „my”, przytrzymaj klawisz Ctrl na klawiaturze i naciśnij spację. Powinieneś otrzymać taki widok:



Edytory zbierają nazwy i zapewniają autouzupełnianie.

VS Code próbuje automatycznie uzupełnić twój zapis przy użyciu już znalezionej nazwy zmiennej: myName. Aby uzyskać nazwę tej zmiennej, TypeScript przeszedł przez twój plik i próbował zebrać wszystkie nazwy, jakie mógł znaleźć, abyś mógł ich ponownie użyć podczas pisania. Mimo że VS Code zapewnia nam TypeScript od razu po wyjęciu z pudełka, możesz mieć te same funkcje w wybranym edytorze. Poszukaj wtyczek o nazwie „Serwer języka TypeScript”. Serwer języka TypeScript implementuje protokół, który może pobierać kod z edytora i zwracać informacje zwrotne dotyczące analizy. Daje to czerwone faliste linie w prawie każdym dostępnym edytorze. Więc TypeScript jest aktywny, ale jest też trochę nieśmiały. Aby uwolnić pełną moc TypeScriptu, musimy wyraźnie poprosić TypeScript o przekazanie nam wyników analizy. Dodając //@ ts-check jako pierwszą linię pliku,

TypeScript zacznie dodawać czerwone faliste linie do fragmentów kodu, które po prostu nie mają sensu:

```
//@ts-check

const storage = {

max: undefined,

items: []

}

Object.defineProperty(storage, 'max', { readonly: true,

val: 5000 })

let currentStorage = 'undefined'

function storageUsed() {

if(currentStorage) {

return currentStorage

}

currentStorage = 0

for(const i = 0; i < storage.length(); i++) {

currentStorage += storage.items[i].weigth

}

return currentStorage

}

function add(item) {

if(storage.max - item.weight >= storageUsed) {

storage.items.add(item)

currentStorage += iten.weight

}

}
```

Och, wow! Wygląda na to, że jest kilka problemów! Niektóre z nich wydają się oczywiste, inne nie. Zbadajmy je jeden po drugim.