Lekcja 6: Pliki deklaracji otoczenia

Dostosujmy naszą funkcję addItem do małego szczegółu: podczas programowania chcemy rejestrować bieżącą ilość w naszym magazynie na konsoli. Debugowanie staje się łatwiejsze, ponieważ stale otrzymujemy informacje zwrotne na temat zmian w pamięci masowej. Aby mieć pewność, że logujemy się tylko podczas programowania, tworzymy globalną flagę isDevelopment, która może przyjmować wartość typu boolean. Jeśli ta wartość jest ustawiona na true, rejestrujemy; w przeciwnym razie nie. Podczas programowania dołączamy nasz JavaScript do specjalnie przygotowanego pliku HTML, który ustawia tę konkretną flagę na true:

<script>

const isDevelopment = true

</script>

W naszym produkcyjnym kodzie HTML pomijamy ten fragment lub ustawiamy isDevelopment na false:

<script>

const isDevelopment = false

</script>

Mamy teraz globalnie zdefiniowaną stałą, do której możemy łatwo uzyskać dostęp z dowolnego miejsca w naszym skrypcie. W zależności od środowiska, w którym dołączamy nasze pliki JavaScript, otrzymujemy inny wynik. Nasza dostosowana funkcja addItem wygląda mniej więcej tak:

```
/**
* @param {StorageItem} item
*/
function add(item) {
if(storage.max - item.weight >= storageUsed()) {
storage.items.push(item)
currentStorage += item.weight
}
if(isDevelopment) {
const itemCount = storage.items.length
console.log(`${itemCount} items`)
console.log(`${currentStorage} kg total`)
}
}
```

To bardzo powszechny wzorzec. A jeśli piszesz zwykły JavaScript, prawdopodobnie polegałeś na jakiejś globalnej stałej lub zmiennej. Ale TypeScript zgłasza błędy! Czerwone zygzaki poniżej isDevelopment,

ponieważ isDevelopment nie jest nigdzie zdefiniowane. TypeScript nie może znaleźć tej konkretnej nazwy.

Niestandardowe deklaracje otoczenia

A TypeScript ma rację! Samo istnienie isDevelopment zależy od naszej dobrej woli. TypeScript nie mógł dowiedzieć się, czego się spodziewać. Czy jest to wartość logiczna, ciąg znaków, obiekt złożony, funkcja czy tylko niezdefiniowana? W tej chwili wiemy, że jest to wartość logiczna. Ale czy nasi współpracownicy za trzy miesiące do przodu? Aby uczynić globalne znane i zdefiniowane, możemy użyć deklaracji typu otoczenia. Te typy obejmują, istnieją i są obecne ze wszystkich stron. Potrzebujemy innego pliku .d.ts do umieszczenia gdzieś w pobliżu naszych typów, gdzie możemy zdefiniować nagłówki funkcji, obiekty globalne i zmienne, których potrzebujemy w całym programie. Utwórzmy plik ambient.d.ts obok naszego głównego pliku JavaScript. Dodamy tam jedną linię:

declare const isDevelopment: boolean

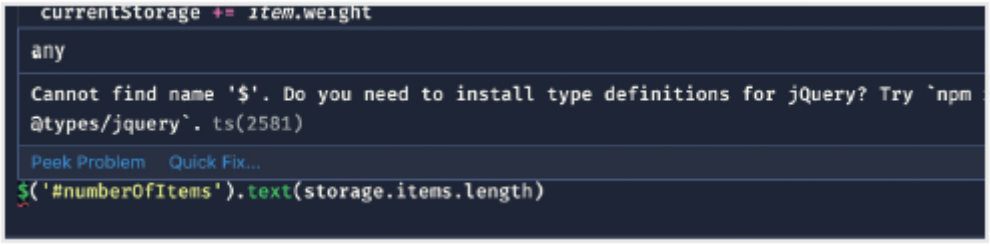
Znów jesteśmy na terytorium języka TypeScript. I znowu, widać pewne podobieństwa do JavaScript. const isDevelopment to część pobrana bezpośrednio z JavaScript. Słowo kluczowe replace przed nim mówi TypeScript, że chcemy poinformować, że ta stała istnieje. Ponieważ jest to tylko deklaracja, nie musimy dodawać konkretnych wartości. Część po const isDevelopment to adnotacja typu i mały spoiler do Części 2. Za pomocą: boolean mówimy TypeScript, że isDevelopment jest - no cóż - logiczne! Zapisz tę linię, a isDevelopment stanie się dostępny we wszystkich twoich plikach JavaScript.

Instalowanie deklaracji biblioteki otoczenia

Deklaracje otoczenia są przydatne nie tylko w przypadku flag globalnych, ale także wtedy, gdy masz funkcje i obiekty, których istnienia oczekujesz. Jednym z przykładów może być jQuery! Tak, jedyna biblioteka JavaScript, która szturmem podbiła tworzenie stron internetowych. I który jest nadal szeroko używany przez miliony, ale w ciągu ostatnich kilku lat w jakiś sposób stracił łaskę. Ale hej, wiele osób zawdzięcza swoją karierę jQuery. Okażmy więc trochę wdzięczności i wykorzystajmy to do celów demonstracyjnych w naszym prostym programie do przechowywania danych. Załóżmy, że chcemy aktualizować liczbę pozycji wyświetlanych w sieci za każdym razem, gdy dodajemy nową pozycję do magazynu naszego statku. Mały fragment kodu jQuery w addItem załatwi sprawę:

```
/**
* @param {StorageItem} item
*/
function add(item) {
if(storage.max - item.weight >= storageUsed()) {
storage.items.push(item)
currentStorage += item.weight
}
$('#numberOfItems').text(storage.items.length)
}
```

Ponownie otrzymujemy ten sam błąd: TypeScript can’t find $, skrótu jQuery. Ale TypeScript jest również bardzo sprytny. Dodawanie jQuery jest bardzo powszechne, więc TypeScript już podpowiada, jak instalować typy jQuery:



Pojawienie się $ zachęca TypeScript do zasugerowania instalacji typów jQuery

TypeScript ma dużą społeczność ludzi, którzy wnoszą niestandardowe typy do prawie każdej biblioteki, która nie została napisana w języku TypeScript, ale mogłaby skorzystać z informacji o typach. Ponieważ jQuery jest nadal szeroko stosowany, była jedną z pierwszych bibliotek, które otrzymały to specjalne traktowanie. Wszystkie definicje typów są dostępne w npm, rejestrze pakietów JavaScript. Node.js i npm to kluczowe elementy zestawu narzędzi dla programistów internetowych. Nie będziemy zagłębiać się w szczegóły pracy z Node.js i npm, ale „Get Started With Node: An Introduction to APIs, HTTP And ES6 + JavaScript” autorstwa Jamiego Corkhill'a w Smashing Magazine zawiera o tym informacje. Więc otwórz swój ulubiony terminal (wskazówka: VS Code ma jeden zintegrowany) i zainstaluj typy jQuery przez npm w folderze twojego aktualnego projektu:

npm i @ types / jquery

I właśnie w ten sposób otrzymujesz pełne automatyczne uzupełnianie i sprawdzanie typów jQuery w plikach JavaScript. Deklaracje typu jQuery są otoczone i TypeScript może z nimi współpracować. Musisz upewnić się, że jQuery istnieje podczas wdrażania aplikacji.