Lekcja 7: Narzędzia

Dodanie języka TypeScript do przepływu pracy programowania zostało zaprojektowane tak, aby było jak najmniej uciążliwe. Jeśli masz edytor skoncentrowany na języku TypeScript, taki jak VS Code, dodawanie typów i uzyskiwanie obsługi edytora po prostu działa. Nie ma potrzeby konfiguracji, ustawień projektu ani żadnych innych narzędzi. Dopiero w Lekcji 6 musieliśmy w końcu coś zainstalować: deklaracje typów bibliotek zewnętrznych, które pomagają nam pracować z jQuery. Dzieje się tak, ponieważ VS Code przyjmuje wiele założeń dotyczących struktury projektu i plików, z którymi pracujesz. Jest idealnie dopasowany, aby ułatwić rozpoczęcie pracy. I to jest zrozumiałe. Zespół VS Code rozpoczął działalność w tym samym czasie, co zespół TypeScript, mając wspólne cele. VS Code został zaprojektowany jako idealny edytor TypeScript.

tsc

Ale co, jeśli potrzebujesz TypeScript poza VS Code? JeśliTwoi współpracownicy używają Sublime Text, Atom, VIM lub czegoś zupełnie innego? Część narzędziowa języka TypeScript jest również dostępna poza programem VS Code. Możemy zainstalować narzędzie wiersza poleceń TypeScript globalnie na naszym komputerze:

npm install -g typescript

Dzięki temu otrzymujemy narzędzie o nazwie tsc, kompilator TypeScript. Podstawowym zadaniem kompilatora TypeScript jest pobranie kodu TypeScript i skompilowanie go do zwykłego JavaScript. Ale dzięki tsc możemy również sprawdzać typy naszych programów JavaScript poza dowolnym edytorem. Jak w przypadku każdego kompilatora, należy ustawić wiele flag i konfiguracji. Przejdźmy do folderu głównego naszego projektu. Tworzymy folder o nazwie @types i przenosimy do niego ambient.d.ts i types.d.ts. Następnie uruchamiamy następujące polecenie w naszym terminalu:

tsc --init

Po zainicjowaniu nowego projektu otrzymujemy wstępnie wypełniony plik tsconfig.json. To jest główny plik konfiguracyjny dla języka TypeScript. Redaktorzy podnoszą go i zachowują się odpowiednio, a kiedy uruchamiasz tsc w terminalu, TypeScript również odnosi się do tego pliku.

tsconfig.json

Wstępnie wypełnione wartości w tsconfig.json są całkiem dobre, gdy chcesz rozpocząć od projektu TypeScript. Chcemy jednak sprawdzać typowo JavaScript, więc musimy wprowadzić kilka poprawek. Tak powinien wyglądać plik tsconfig.json:

```
{

"compilerOptions": {

"target": "ES2020",

"moduł": "es2020",

"allowJs": true,

"checkJs": true,

"typeRoots": [

"@types",
```

„node_modules / @ types”

],

"esModuleInterop": true,

}

}

Istnieje kilka ustawień i flag, które są wstępnie ustawione i są bardzo przydatne po skompilowaniu kodu TypeScript do kodu JavaScript:

1. target: Po skompilowaniu języka TypeScript do JavaScriptu musimy zdefiniować specyfikację języka docelowego. Może to być jeden z najnowszych lub aktualnych standardów ECMAScript (na którym bazuje standardowy JavaScript), na przykład ES3 dla starszych przeglądarek, ES5 dla starszych środowisk wykonawczych lub wszystko, od ES2015 do tegorocznej specyfikacji. Jeśli zawsze wybierasz najnowsze i najlepsze, użyj ESNEXT.

2. modul. Kolejna flaga kompilatora, która jest ważna, gdy pracujemy z modułami. Jeśli importujemy i eksportujemy w TypeScript, jak powinny być traktowane przez język docelowy, do którego kompilujemy? Czy jest to commonjs (używany przez Node.js), czy es2020 (system modułów nowoczesnej przeglądarki), czy jeden z wielu, które są w pobliżu?

3. esModuleInterop. Mocno połączony z modułem powyżej. Jeśli chcesz mieszać moduły z różnych systemów modułów, takich jak ES Modules i CommonJS, możesz ustawić tę flagę na true, a TypeScript zajmie się kompatybilnością.

Być może nie będziemy ich teraz potrzebować, ale użyjemy ich później. Inne opcje kompilatora są szczególnie przydatne podczas sprawdzania typu JavaScript:

1. allowJs. Ta flaga mówi TypeScript, aby zezwolił na odwołanie do zwykłych plików JavaScript.

2. checkJs. Ta flaga jest podobna do komentarza // @ ts-check, którego używaliśmy wcześniej. Mówi TypeScript, aby sprawdzał pliki JavaScript.

3. typRoots. Tutaj mówimy kompilatorowi TypeScript, co pierwotnie zrobił VS Code: foldery, w których można znaleźć dodatkowe informacje o typie. Jednym z nich jest nasz lokalny folder @types, w którym ładujemy deklaracje modułów otoczenia. Drugi to node_modules / @ types, z którego pobieramy typy jQuery.

Automatycznie utworzony plik tsconfig.json jest bardzo dobrze udokumentowany. Dla każdej flagi widzimy wartości domyślne i komentarz obok niej wyjaśniający, do czego służy ta flaga. Ponadto podczas pracy z VS Code otrzymujemy fajne funkcje automatycznego uzupełniania, jeśli chcemy ustawić i zmienić ustawienia. Teraz, gdy kompilator TypeScript jest skonfigurowany, pora na sprawdzenie typu. Uruchommy następującą komendę w naszym terminalu:

tsc --noEmit

tsc pobierze ustawienia z tsconfig.json. Parametr --noEmit mówi TypeScript, że chcemy tylko sprawdzić typy i nie tworzyć żadnych plików wyjściowych.

Jeśli wszystko w naszym kodzie jest w porządku, tsc kończy działanie bez błędu i ostrzeżenia. Jeśli chcesz regularnie sprawdzać typ kodu podczas programowania, możesz dodać tryb obserwacyjny, aby TypeScript ponownie sprawdzał typ za każdym razem, gdy zapisujesz plik.

tsc --noEmit --watch

I dzięki temu skonfigurowaliśmy odpowiednie narzędzia, które pozwalają nam sprawdzać kod JavaScript poza edytorem, co jest dobrą podstawą dla wszystkich przyszłych przykładów TypeScript!

Podsumowanie

W tej części mieliśmy delikatne wprowadzenie do języka TypeScript z perspektywy narzędzi. Naszym głównym celem było umieszczenie w kodzie czerwonych falistych linii za każdym razem, gdy coś wydawało się podejrzane! Korzystając z edytora opartego na TypeScript, takiego jak VS Code, byliśmy w stanie aktywować coraz więcej funkcji krok po kroku:

1. Dodanie // @ ts-check dało nam wstępne wyobrażenie o tym, co jest nie tak z naszym kodem. TypeScript używa koncepcji zwanej wnioskiem o typie, aby automatycznie wykrywać typy stałych, funkcji i zmiennych.

2. Korzystając z komentarzy JSDoc, mogliśmy tworzyć niestandardowe typy i dodawać adnotacje do typów w całym naszym kodzie, aby umożliwić TypeScript jeszcze lepsze zrozumienie intencji naszego programu.

3. Słyszeliśmy, że TypeScript jest strukturalnym systemem typów, co oznacza, że TypeScript bardzo dba o kształt lub strukturę obiektów i funkcji.

4. W plikach deklaracji typów pisaliśmy nasze pierwsze typy w języku programowania TypeScript i odnosiliśmy się do nich w komentarzach JSDoc.

5. Deklaracje typu otoczenia pozwoliły nam ustawić informacje o typie dla globalnych, takich jak jQuery lub niestandardowe flagi środowiska.

6. Na koniec zainstalowaliśmy i skonfigurowaliśmy kompilator TypeScript, więc wszystkie niejawne ustawienia z edytora zostały zapisane i można je było zmienić. Pozwoliło nam to uzyskać takie same wyniki u wszystkich redaktorów i spoza nich.

Teraz, gdy zrobiliśmy już pierwsze kroki z językiem TypeScript i trochę zaznajomiliśmy się ze sposobem, w jaki język TypeScript wpływa na nasz przepływ pracy w zakresie kodowania, czas zająć się typami w następnej części!