Lekcja 8: Kompilowanie TypeScript

W tej części zamierzamy dodać kilka drobiazgów, które pomogą nam stworzyć sklep internetowy: funkcje narzędziowe do kasy, kilka dynamicznych nakładek i trochę komunikacji z zapleczem. Ale tym razem chcemy wyjść z naszej strefy komfortu i napisać rzeczywisty TypeScript. A ponieważ TypeScript nie może być wykonany przez przeglądarkę, musimy skompilować go do zwykłego JavaScript.

Konfigurowanie kompilatora

Aby rozpocząć od miejsca, w którym zakończyliśmy w poprzednim przykładzie, robimy trzy rzeczy:

1. Zmień nazwę naszego głównego pliku JavaScript tak, aby kończyła się na .ts. To jest teraz plik TypeScript!

2. Utwórz nowy plik example-two.ts, do którego będziemy mogli dodać wszystkie przykłady z tej części.

3. Dostosuj nasz plik tsconfig.json, aby ignorował JavaScript.

{

"compilerOptions": {

"target": "ES2020",

"moduł": "es2020",

"typeRoots": [

"@types",

„node_modules / @ types”

],

"esModuleInterop": true,

}

}

Moment, w którym biegniemy

tsc

w naszym terminalu widzimy, że pojawiają się dwa nowe pliki: skompilowany plik JavaScript dla każdego pliku TypeScript. Jeśli je otworzysz, zobaczysz, że prawie nic się nie zmieniło w porównaniu z naszymi oryginalnymi plikami TypeScript. Dzieje się tak, ponieważ nie zrobiliśmy nic związanego z TypeScript. Jeszcze. Ponieważ TypeScript jest nadzbiorem języka JavaScript, cały kod JavaScript jest kodem TypeScript. Ale ponieważ TypeScript jest nadzbiorem, język to nie wszystko.

Nasze adnotacje pierwszego typu

W example-two.ts tworzymy funkcję użyteczności do dodania podatku od towarów i usług (VAT) do zwykłej ceny produktu. Cena jest podana w wybranej przez Ciebie walucie; VAT jest wartością procentową wyrażoną jako ułamek dziesiętny (np. 20% to 0,2).

function addVAT(price, vat) {

return price * (1 + vat)

}

Od razu włącza się wnioskowanie o typie w języku TypeScript. Patrząc na obliczenia w treści funkcji, TypeScript wie, że mamy do czynienia z liczbami (operator mnożenia i cyfra 1 sugerują, że):

```
// vatPrice jest typu ‘number’

const vatPrice = addVAT(30, 0.2)
```

Jednak nadal możemy przekazywać jako parametry funkcji wszystko oprócz liczb:

```
/ vatPrice jest typu ‘number’

const vatPriceWrong = addVAT('this is so', 'wrong')
```

Wartość vatPriceWrong to NaN, co oznacza „not a number”. Co zabawne, NaN jest typu liczbowego, ponieważ może wynikać tylko z operacji na typie liczbowym, ale z jedną lub wieloma wartościami nie o typie liczbowym. Z technicznego punktu widzenia TypeScript jest więc poprawny. Nasze oprogramowanie jest poprawne. Ale chcemy być lepsi niż to. Chcemy mieć pewność, że nie otrzymamy wartości, z którymi nie chcemy mieć do czynienia. Jednym ze sposobów jest dodanie domyślnej wartości vat:

```
function addVAT(price, vat = 0.2) {

return price * (1 + vat)

}
```

Teraz TypeScript może ponownie wywnioskować:

```
const vatPrice = addVAT(30, 0.2) // OK!

const vatPriceWithDefault = addVAT(30) // OK!

// Nie OK. Oczekujemy liczby dla vat ze względu na

// domyślną wartość! Ten kawałek powoduje błędy

const vatPriceErrors = addVAT(30, 'a string!')

// Nie jest to jednak całkiem rozsądne, ale OK

const vatPriceAlsoWrong = addVAT('Hi, friends!')
```

Oto, jak daleko możemy się posunąć za pomocą wnioskowania o typie. Jeśli spojrzymy na naszą funkcję addVAT, zobaczymy:

• Wartość zwracana jest typu liczba ze względu na rodzaj operacji wewnątrz funkcji.

• vat jest typu number, ponieważ wartością domyślną jest liczba.

• cena jest typu any.

any jest specjalnym typem TypeScript - nie istnieje w JavaScript. Akceptuje dowolną wartość dowolnego typu, a zatem jest typem najwyższym, obejmującym wszystkie inne typy. TypeScript ustawia dowolny typ jako domyślny dla dowolnej wartości lub parametru, który nie został jawnie wpisany lub którego nie można wywnioskować.

Aby być jeszcze bardziej wyraźnym i celowym w naszych typach, musimy dodać adnotacje typu. W części 1 dodaliśmy adnotacje typu poprzez komentarze JSDoc. Teraz możemy to zrobić bezpośrednio w naszej głowie funkcji, gdzie to się dzieje:

function addVAT(price: number, vat = 0.2) {

return price * (1 + vat)

}

Zauważ, że umieściliśmy cenę jako numer typu. Widzieliśmy coś podobnego w naszych plikach .d.ts z Części 1. Dzięki temu TypeScript doda piękne czerwone faliste linie za każdym razem, gdy przekażemy parametr, który nie działa.

const boom = addVAT('this is not a number!')

Zamiast dodawać adnotacje typu w komentarzach, dodajemy je bezpośrednio do deklarowanych nazw i parametrów. Powyższe definicje typów są krótkimi formami:

function addVAT(price: number, vat: number = 0.2):

number {

return price * (1 + vat)

}

Tutaj jesteśmy jeszcze bardziej wyraźni. Deklarujemy typy dla obu parametrów, a nawet typ zwracany (ostatnia liczba przed nawiasem klamrowym).

Taka jawność ma specjalny wpływ na TypeScript: Type-Script nie wnioskuje już o typach - sprawdza, czy domyślna wartość vat i zwracana wartość ceny * (1 + vat) odpowiadają typom zadeklarowanym w nagłówku funkcji.

Kompilacja do JavaScript

Jednak środowisko wykonawcze JavaScript przeglądarki nie może uruchomić języka TypeScript, więc musimy pozbyć się wszystkich adnotacji. Uruchom ponownie tsc w swoim terminalu. TypeScript weźmie nasz nowy plik example-two.ts, utworzy plik example-two.js, a zawartość będzie taka sama, bez adnotacji wszystkich typów - JavaScript, który możesz uruchomić w przeglądarce.

Zwróć uwagę, że po etapie kompilacji po uruchomieniu kodu tracisz bezpieczeństwo typów: jest to zwykły JavaScript, gdy trafi on do przeglądarek. Jeśli ktoś wywoła funkcję addVAT poza twoją aplikacją, nadal może ją uruchomić z parametrami różnych typów, potencjalnie powodując awarię aplikacji. Jeśli udostępniasz swoje funkcje jako API światu zewnętrznemu, zawsze dobrze jest przeprowadzić dodatkowe kontrole typu i odpowiednią obsługę błędów.

Możesz także uruchomić tsc w trybie oglądania, aby otrzymywać regularne aktualizacje po zapisaniu plików:

tsc --watch

Jeśli chcesz, możesz na tym etapie pobawić się niektórymi opcjami kompilatora. Ustaw cel w swoim tsconfig.json na inną wartość (np. Es5) i zobacz, jak wygląda skompilowane dane wyjściowe po zmianie ustawień. TypeScript nie tylko usuwa wszystkie typy, ale także wykorzystuje współczesne funkcje JavaScript i transponuje je do starszych wersji ECMAScript. TypeScript, który generuje JavaScript,

nazywa się emitowaniem. Pamiętasz flagę --noEmit z Części 1? To pozwala nam sprawdzać typy bez emitowania dodatkowych plików. Jeśli ta flaga nie jest ustawiona (domyślnie), zawsze otrzymujemy JavaScript - nawet jeśli sprawdzenie typu nie powiedzie się. Jeśli chcemy się upewnić, że nie otrzymamy żadnych wyemitowanych danych wyjściowych JavaScript, ustaw noEmitOnError na true w tsconfig.json.