Elektrotehnički fakultet Sarajevo

i

ZIRA Sarajevo

Završni rad - praksa

Sprint 1 – Typescript

Sarajevo, 7.5.2020.

1 Uvod

Typescript se instalira koristeći Node paket manager sljedećom komandom.

```
npm install -g typescript
```

Typescript predstavlja ekstenziju na Javascript te podržava sav Js kod kao i dodatne mogućnosti poput statičnih tipova. Ova znači da je moguće pisati i NodeJS backend kroz Typescript.

1.1 Include

Da bi koristili npr. NodeJS, potrebno je da uključimo paket koji definište sve Node tipove. To radimo na 2 načina. Prvi je sa reference tagom na mjestu gdje želimo importovati.

```
/// <reference path="node.d.ts" />
```

Drugi način je kroz tsconfig.json file koji definiše parametre prilikom kompajliranja. Sve datotoke koje opisuju tipove stavljamo pod typeRoots tag. Moguće je postaviti dodatne opcije prilikom kompajliranja poput direktorija u kojem će se nalaziti obični Javascript kod koji moža ne želimo da miješamo sa Typescriptom u istom direktoriju.

```
{
    'compilerOptions': {
        'target': 'es5',
        'module': 'commonjs',
        'sourceMap': true,
        'outDir': 'out',
        'typeRoots': [
            './src/node.d.ts'
        ]
    }
}
```

Statički tipovi se definiraju sa dodavanjem :tip poslije imena varijable. Typescript podržava i Javascript način deklaracije bez implicitnog zadavanja tipa. Varijable koje su deklarisane sa declare se mogu koristiti na nivou cijele aplikacije. Varijable čiji tip još nije definiran su tipa any. Od svih tipova je moguće kreirati nizove.

```
declare var temp;
var tekst: string = 'primjer';
var tekst2 = 'js way';
var nepoznato: any;
var names: string[] = ['John'];
var kolicina: boolean = null;
var house: number = undefined;
```

Null tip je podtip svih primitivnih tipova osim void i undefined. Undefined tip je podtip svih tipova. Sve varijable koje nisu inicijalizovane su po default-u undefined.

Također je moguće specificirati tipove ulaznih parametara u funkcije.

```
function sum(num: number, text: string, optional?: string): string {
```

```
    return num + text + optional;
}
```

Prilikom poziva funkcije, sve proslijeđene varijable moraju se poklapati sa očekivanim ulaznim tipovima. Pored toga, nije moguće poslati vrijednost u funkciju koja ne očekuje nikakvu vrijednost za razliku od Js-a koji samo ignorira proslijeđenu vrijednost. Opcionalni parametri se označavaju sa „?“ za razliku od Js-a gdje je svaki parametar opcionalan. Dodatno, provjera tipova se vrši prilikom kompajliranja umjesto prilikom rada kod Js-a.

```
var mult = (h: number, w: number) => h * w;
var update : (h: number) => void;
```

Brži način pisanja return-a je iza „=>“ znaka. Ukoliko funkcija ne vraća vrijednost, onda se piše void kao povratni tip.

1.2 Interfejsi

U Typescriptu i funkcije mogu implementirati interfejs ili više njih istovremeno. Kao povratni tip iz funkcije mora biti objekat sa svim atributima interfejsa odnosno implementacijom svih funkcija interfejsa.

```
interface sessionEval {
    addRating: (rating: number) => void;
    calcRating: () => number;
}

function sessionEvaluator(): sessionEval {
    var ratings: number[] = [];
    var addRatingImpl = (rating: number) => {
        ratings.push(rating);
    }
    var calcRatingImpl = () => {
        var sum: number = 0;
        ratings.forEach(function (el) {
            sum += el;
        });
        return sum / ratings.length;
    }

    return {
        addRating: addRatingImpl,
        calcRating: calcRatingImpl
    }
}
```

1.3 Klase

Svi atributi klase su automstski public osim ako nije naglašeno da su private. Ako su atributi koje prima construktor public, onda će ih Typescript automstski kreirati i inicijalizirati te nije

potrebno da ih ručno definišemo. Privatni atribut je potrebno nazvati drugačije od gettera i settera jer će prilikom poziva this.atribut se pozvati setter a ne direktno atribut.

```
class Engine {
    constructor(public horsePower: number, public engineType: string) { }
}

class Car {
    private _engine: Engine;

    constructor(engine: Engine) {
        this.engine = engine;
    }

    get engine(): Engine {
        return this._engine;
    }

    set engine(value: Engine) {
        if (value == undefined) throw 'Please supply an engine';
        this._engine = value;
    }

    start() : void {
        alert('Car engine started ' + this._engine.engineType);
    }
}
```

1.4 Konverzija tipova

Tipove je moguće pretvoriti u druge uz dodavanje <Tip> ispred vrijednosti koju želimo konvertovati. U primjeru ispod, getElementById vraća tip HTMLElement te ga je potrebno konvertovati u HTMLInputElement sa <HTMLInputElement>. Konverzija je jako korisna kod pretvorbe baznog tipa u izvedeni tip kada smo sigurni da je izvedeni tip sačuvan u baznom (slično polimorfizmu).

```
var x: HTMLInputElement = <HTMLInputElement>document.getElementById("xId");

var engine: IEngine = new Engine(100, '1.6 CDI');
console.log((<Engine>engine).horsePower); //konverzija je neophodna
```

1.5 Rest parametar

Typescript omogućava korištenje rest parametara koji grupišu više proslijeđenih vrijednosti istog tipa u jednu kolekciju.

```
addAccessories(...accessories: Accessory[]) {
    this.accessoryList = '';
    for (var i = 0; i < accessories.length; i++) {
        var ac = accessories[i];
        this.accessoryList += ac.accessoryNumber + ' ' + ac.title + '<br />';
```

```
    }
}
```

1.6 Moduli

Moduli služe za razdvajanje i grupisanje koda. Interni moduli se deklarišu sa namespace, a eksterni sa module. Sve što želimo da bude vidljivo van modula eksportujemo sa export. Dodavanje reference na druge module se radi u reference tagu na početku dokumenta. Ova referenca služi samo za IntelliSense i kompajliranje te neće referencirati generisani Javascript kod što znači da moramo paziti na redoslijed kojim uključujemo Javascript datoteke u HTML.

```
namespace App.Shapes {
    export interface IPoint {
        getDist(): number;
    }
}
```

Eksterne module exportujemo sa export direktno bez namespace-a, a importujemo sa require.

```
import ds = require('./dataservice');

export interface IAlerter {
    name: string;
    showMessage(): void;
}
```

Koristeći RequireJS rješavamo problem redoslijeda uključivanja Js-a.