Nathan Rozentals

Naučite TypeScript

PREVOD DRUGOG IZDANJA

Napravite poslovne i industrijske veb aplikacije pomoću TypeScripta i vodećih JavaScript okruženja

kompjuter biblioteka

Packt

Nathan Rozentals

Naučite TypeScript

Prevod II izdanja

Izgradite poslovne veb aplikacije
industrijske jačine, koristeći TypeScript
i vodeće JavaScript radne okvire

kompjuter biblioteka

Packt>

Izdavač:

kompjuter biblioteka

Obalskih radnika 15, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Nathan Rozentals

Prevod: Slavica Prudkov

Lektura: Miloš Jevtović

Slog : Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2017.

Broj knjige: 491

Izdanje: Prvo

ISBN: 978-86-7310-514-7

Mastering TypeScript
Second Edition

by Nathan Rozentals

ISBN 978-1-78646-871-0

O AUTORU

Nathan Rozentals gradi komercijalni softver već više od 26 godina, a programira u mnogo dužem periodu. Pre nego što je Internet postao popularan, on je gradio programe za statističke analize za velike računarske sisteme. Kao i mnogi programeri tada, pomogao je u „spašavanju sveta" 2000. godine.

Ovladao je mnogim objektno-orijentisanim jezicima, počevši od implementiranja objektno-orijentisanih tehnika u „dobri stari" jezik C. Proveo je više godina koristeći C++, pokušavajući da reši probleme zaključavanja niti i rekurzivnih rutina koje izazivaju „curenje" memorije, pa je odlučio da pojednostavi svoj život i prihvati automatsko sakupljanje otpadaka u jeziku Java, a zatim i u jeziku C#.

Kada su veb tehnologije postale popularne, fokusirao se na moderno veb programiranje, pa, samim tim i na JavaScript. U TypeScriptu je pronašao jezik u kojem može u JavaScriptu da iskoristi sve objektno-orijentisane obrasce koje je naučio tokom godina.

Da nije bilo ekstremnih tehnika programiranja, agilnog isporučivanja, razvoja vođenog testiranjem koda i kontinuirane integracije, on bi „izgubio razum" pre mnogo godina.

Kada ne programira, on razmišlja o programiranju. Da bi prestao da razmišlja o programiranju, ide na jedrenje na dasci, igra fudbal ili, jednostavno, gleda profesionalce koji igraju fudbal.

O RECENZENTIMA

Guy Fergusson je strastveni veb programer, saradnik u zajednici otvorenog koda i gejmer. Gradio je aplikacije u sektoru zdravstva, prava i finansija. Radio je sa autorom na izgradnji TypeScript aplikacija i sada je pristalica TypeScripta, koji svakodnevno koristi.

Vilic Vane je JavaScript inženjer sa više od osam godina iskustva u veb razvoju. Počeo je da prati TypeScript projekat kada je izdat i takođe je sarađivao pri njegovoj izradi. Sada radi na radnim okvirima, bibliotekama i aplikacijama napisanim u TypeScriptu. Vilic je autor knjige „TypeScript Design Patterns".

Kratak sadržaj

Sadržaj

POGLAVLJE 2

POGLAVLJE 3

POGLAVLJE 8

POGLAVLJE 9

POGLAVLJE 10

UVOD

Jezik TypeScript i kompajler, koji su se pojavili krajem 2012. godine, i dalje su veoma uspešni. Veoma brzo su urezali čvrst trag u zajednici JavaScript razvoja i nastavljaju da budu sve jači i jači. Za mnoge velike JavaScript projekte, uključujući projekte Adobe, Mozilla i Asana, odlučeno je da prebace osnovu koda iz JavaScripta u TypeScript. Nedavno su timovi „Microsofta“ i „Googlea“ objavili da će Angular 2.0 biti razvijen pomoću TypeScripta, čime će jezici AtScript i TypeScript biti spojeni u jedan.

Ovolika popularnost TypeScripta pokazuje vrednost ovog jezika, fleksibilnost kompajlera i poboljšanja u produktivnosti koji se mogu realizovati upotrebom bogatog seta razvojnih alata ovog jezika. Osim ove industrijske podrške, standardi ECMAScript 6 i ECMAScript 7 su sve bliže publikovanju, a TypeScript obezbeđuje način da se upotrebe funkcije ovih standarda u aplikacijama generisanjem kompatibilnog JavaScripta.

Pisanje jednostranih JavaScript aplikacija u TypeScriptu je sada čak i jednostavnije upotrebom velike kolekcije fajlova deklaracije koje su izgradili članovi TypeScript zajednice. Ovi fajlovi deklaracije lako integrišu veliki raspon postojećih JavaScript radnih okvira u razvojno okruženje TypeScripta i time poboljšavaju produktivnost, omogućavaju rano otkrivanje grešaka i napredne IntelliSense funkcije.

Međutim, jezik JavaScript nije ograničen na veb pretraživače. Sada možemo da pišemo JavaScript na serveru, da pokrećemo aplikacije na mobilnom telefonu, koristeći JavaScript, i čak da pomoću JavaScripta kontrolišemo mikro uređaje koji su namenjeni za Internet of Things.

Ova knjiga je vodič i za iskusne TypeScript programere i za one programere koji tek počinju svoje TypeScript „putovanje“. Fokusiranje na razvoj vođen testiranjem koda, detaljne informacije o integraciji sa mnogim popularnim JavaScript bibliotekama i detaljni pregled funkcija TypeScripta u ovoj knjizi pomoći će vam da istražite sledeći korak u JavaScript razvoju.

ŠTA OBUHVATA OVA KNJIGA

U Poglavlju 1, „*TypeScript – alatke i opcije radnog okvira*“, postavili smo scenu za počinjanje TypeScript razvoja. Opisani su prednosti upotrebe TypeScripta kao jezika i kompajlera i podešavanje kompletnog razvojnog okruženja korišćenjem velikog broja popularnih IDE-ova.

Poglavlje 2, „*Tipovi, promenljive i tehnike funkcije*“, predstavlja čitaocu jezik TypeScript, počevši od osnovnih tipova i objašnjenja tipa, a zatim su opisane promenljive, funkcije i napredne funkcije jezika.

Poglavlje 3, „*Interfejsi, klase i nasleđivanje*“, nastavak je teme iz prethodnog poglavlja. U njemu su predstavljeni objektno-orijentisani koncepti i mogućnosti interfejsa, klasa i nasleđivanja. Zatim su ovi koncepti prikazani u radu pomoću Factory Design Patterna.

U Poglavlju 4, „*Dekoratori, generički tipovi i asinhrone funkcije*“, opisane su naprednije funkcije jezika dekoratori i generički tipovi, pre opisa koncepata asinhronog programiranja. Prikazano je kako jezik TypeScript podržava ove asinhrone funkcije kroz deklaracije i upotrebu strukture async await.

Poglavlje 5, „*Pisanje i upotreba fajlova deklaracije*“, provešće čitaoca kroz izgradnju fajla deklaracije za postojeću osnovu JavaScript koda, a zatim su izlistane neke najčešće upotrebljavane sintakse upotrebljene prilikom pisanja fajlova deklaracije. Ove sintakse su namenjene da budu brz referentni vodič za sintaksu fajla deklaracije, ili „puškice“.

U Poglavlju 6, „*Nezavisne biblioteke*“, prikazano je čitaocu kako da upotrebi fajlove deklaracije iz skladišta DefinitelyTyped unutar radnog okruženja. Zatim je opisano kako se piše TypeScript kod koji je kompatibilan sa tri popularna JavaScript radna okvira, čiji su nazivi Backbone, Angular 1 i ExtJs.

U Poglavlju 7, „*Radni okviri kompatibilni sa TypeScriptom*“, opisani su popularni radni okviri koji imaju potpunu integraciju TypeScript jezika. Istražena je MVC paradigma, a zatim je upoređeno kako je ovaj obrazac projektovanja implementiran u radne okvire Backbone, Aurelia, Angular 2 i React.

Poglavlje 8, „*Razvoj vođen testiranjem koda*“, započinje opisom šta je razvoj vođen testiranjem koda, a zatim provodimo čitaoca kroz proces kreiranja različitih tipova testova za kod. Upotreba biblioteke Jasmine prikazuje kako da koristite testove vođene podacima i kako da testirate asinhronu logiku. Poglavlje se završava opisom pokretača testa, izveštaja testa i upotrebe servera kontinualne integracije.

U Poglavlju 9, „*Testiranje radnih okvira kompatibilnih sa TypeScriptom*“, prikazani su testiranje koda i integracije i prihvatanje primera aplikacije koja je izgrađena u svakom radnom okviru koji je kompatibilan sa TypeScriptom. Opisan je koncept mogućnosti testiranja i prikazano je kako male promene u aplikaciji i implementaciji mogu da obezbede daleko bolju pokrivenost testiranja aplikacije.

U Poglavlju 10, „*Modularizacija*“, istraženi su moduli (kako mogu da budu upotrebljeni) i dva tipa generacije modula koje podržava TypeScript kompajler - CommonJs i AMD. Zatim je prikazano kako moduli mogu da budu upotrebljeni pomoću programa za učitavanje modula, uključujući Require i SystemJs. Ovo poglavlje se završava detaljnim opisom upotrebe modula unutar Nodea i izgradnjom primera Express aplikacije.

U Poglavlju 11, „*Objektno-orijentisano programiranje*“, predstavljeni su koncepti objektno-orijentisanog programiranja i prikazano je kako se uređuju komponente aplikacije da bi bili potvrđeni objektno-orijentisani principi. Zatim, sledi detaljan opis primene najbolje prakse objektno-orijentisanog programiranja prikazom kako obrasci projektovanja State i Mediator mogu da se upotrebe za upravljanje složenim interakcijama korisničkog interfejsa.

U Poglavlju 12, „*Dependency Injection*“, opisani su koncepti Service Location i Dependency Injection i način na koji oni mogu biti upotrebljeni za rešavanje uobičajenih problema projektovanja aplikacije. Takođe je prikazano kako se implementira jednostavan radni okvir Dependency Injection pomoću Decoratora.

U Poglavlju 13, „*Izgradnja aplikacija*“, istraženi su osnovni gradivni blokovi razvoja veb aplikacija, uključujući i generisanje HTML stranica iz Nodea i Expressa, pisanje i upotrebu REST krajnjih tačaka i povezivanje podataka. Prikazano je kako se integrišu Express server, REST krajnje tačke i povezivanje podataka pomoću Aurelia, Angulara 2 i Reacta.

U Poglavlju14, „*Prionimo na posao*“, gradimo jednostranu aplikaciju pomoću Angulara 2 i Expressa kombinovanjem svih koncepata i komponenata koji su izgrađeni u knjizi u jednu aplikaciju. Ovi koncepti uključuju razvoj vođen testiranjem koda, obrasce State i Mediator, upotrebu REST krajnjih tačaka, principe objektno-orijentisanog projektovanja, modularizaciju i prilagođene CSS animacije.

ŠTA VAM JE POTREBNO ZA OVU KNJIGU

Biće vam potreban TypeScript kompajler i neka vrsta editora. TypeScript kompajler je dostupan na Windowsu, MacOS-u i Linuxu kao Node plagin. U Poglavlju 1, „*TypeScript – alatke i opcije radnog okvira*“, opisano je podešavanje radnog okruženja.

ZA KOGA JE OVA KNJIGA

Bez obzira da li ste JavaScript programer koji želi da nauči TypeScript ili ste iskusni TypeScript programer koji želi da podigne svoju veštinu na viši nivo, ova knjiga je za vas. Ova knjiga će vam pokazati kako da uključite jake tipove, objektnu orijentaciju i najbolju praksu projektovanja u JavaScript aplikacije.

KONVENCIJE

U ovoj knjizi pronaći ćete veliki broj stilova teksta koji predstavljaju različite vrste informacija. Evo i nekih primera ovih stilova i objašnjanja njihovog značenja.

Reči koda u tekstu, nazivi tabela baze podataka, nazivi direktorijuma, nazivi fajlova, ekstenzije fajla, nazivi putanja, kratki URL-ovi, korisnički unos i Twitter identifikatori su prikazani na sledeći način: „Definisaćemo novu funkciju pod nazivom `MyClass` i vratićemo je u spoljašnju funkciju koja je poziva. Zatim ćemo upotrebiti ključnu reč prototype da bismo injektirali novu funkciju u definiciju funkcije `MyClass`".

Blok koda je postavljen na sledeći način:

```
class MyClass {
  add(x: number, y: number) {
    return x + y;
  }
}
```

Svi unosi ili ispisi komandne linije napisani su na sledeći način:

```
npm  install @types/express
```

Novi termini i **važne reči** su napisani masnim slovima. Reči koje vidite na ekranu, na primer, u menijima ili okvirima za dijalog, biće prikazane u tekstu na sledeći način: „Kliknite na **Run | Debug**, pa promenite konfiguracije. Kliknite na dugme plus (+), selektujte opciju **JavaScript debug** sa leve strane i dodelite naziv konfiguraciji".

Upozorenja ili važne napomene će biti prikazani u ovakvom okviru.

Saveti i trikovi prikazani su ovako.

POVRATNE INFORMACIJE OD ČITALACA

Povratne informacije od naših čitalaca su uvek dobrodošle. Obavestite nas šta mislite o ovoj knjizi – šta vam se dopalo ili šta vam se možda nije dopalo. Povratne informacije čitalaca su nam važne da bismo ubuduće kreirali naslove od kojih ćete dobiti maksimum. Da biste nam poslali povratne informacije, jednostavno nam pošaljite e-mail na adresu feedback@packtpub.com i u naslovu poruke napišite naslov knjige. Ako postoji tema za koju ste specijalizovani ili ste zainteresovani da pišete ili sarađujete na nekoj od knjiga, pogledajte vodič za autore na adresi www.packtpub.com/authors.

KORISNIČKA PODRŠKA

Sada, kada ste ponosni vlasnik „Packt“ knjige, mi imamo mnogo čega da vam ponudimo da bismo vam pomogli da dobijete maksimum iz svoje narudžbine.

Preuzimanje primera koda

Možete da preuzmete fajlove sa primerima koda za ovu knjigu na adresi:

http://www.kombib.rs/preuzimanje/kod/491-naucite-typescript.zip.

Kada su fajlovi preuzeti, ekstrahujte direktorijum koristeći najnoviju verziju:

- WinRAR / 7-Zip za Windows
- Zipeg / iZip / UnRarX za Mac
- 7-Zip / PeaZip za Linux

Paket koda za ovu knjigu možete pronaći i na GitHubu na adresi:

https://github.com/PacktPublishing/Mastering-TypeScript-Second-Edition

PREUZIMANJE SLIKA U BOJI ZA OVU KNJIGU

Takođe smo vam obezbedili kolor snimke ekrana/dijagrama upotrebljenih u ovoj knjizi. Slike u boji će vam pomoći da bolje razumete promene u ispisu. Možete da preuzmete ovaj fajl sa adrese:

http://www.kombib.rs/preuzimanje/kod/naucite-typescript-kolorne-fotografije.rar

ŠTAMPARSKE GREŠKE

Posetite veb stranu knjige: http://knjige.kombib.rs/naucite-typescript-prevod-drugog-izdanja i ostavite komentar. Kada je greška verifikovana, vaša prijava će biti prihvaćena i greška će biti aploudovana na naš veb sajt ili dodata u listu postojećih grešaka, pod odeljkom Greške za konkretni naslov.

PIRATERIJA

Piraterija autorskog materijala na Internetu je aktuelan problem na svim medijima. „Packt“ zaštitu autorskih prava i licenci shvata veoma ozbiljno. Ako pronađete ilegalnu kopiju naših knjiga, u bilo kojoj formi, na Internetu, molimo vas da nas o tome obavestite - pošaljite nam adresu lokacije ili naziv veb sajta da bismo mogli da podnesemo tužbu.

Molimo vas, kontaktirajte sa nama na adresi copyright@packtpub.com i pošaljite nam link ka sumnjivom materijalu.

Unapred smo vam zahvalni na pomoći u zaštiti naših autora i mogućnosti da vam pružimo vredan sadržaj.

1

TypeScript – alatke i opcije radnog okvira

JavaScript je zaista sveprisutni jezik. Skoro svaki veb sajt koji posetite u modernom svetu koristi JavaScript koji ga čini prilagodljivijim, čitkijim ili privlačnijim za upotrebu. Čak se i tradicionalne desktop aplikacije prebacuju na Internet. Nekada je trebalo da preuzmemo i instaliramo program da bismo generisali dijagram ili napisali dokument, a sada sve to možemo da uradimo na Vebu, unutar granica skromnog pretraživača.

To je moć JavaScripta, koji omogućava da preispitamo način na koji koristimo Veb. Ali nam isto tako omogućava da preispitamo način na koji koristimo veb tehnologije. Na primer, Node omogućava pokretanje JavaScripta na strani servera, renderovanje celih veb sajtova velikog obima, završavanje obrade sesije, raspoređivanje opterećenja i interakciju sa bazom podataka. Međutim, ova promena u razmišljanju o veb tehnologijama je samo početni korak u unapređenja veb sajtova.

Apache Cordova je punopravni veb server koji se pokreće kao izvorna aplikacija za mobilni telefon. To znači da možemo da gradimo aplikaciju za mobilni telefon, koristeći HTML, CSS i JavaScript, a zatim da vršimo interakciju sa akcelerometrom telefona, uslugama geografske lokacije ili skladištem fajlova. Stoga, koristeći server Cordova, JavaScript i veb tehnologije uvedene su u domen izvornih aplikacija za mobilne telefone.

Isto tako, projekti kao što je *Kinoma* koriste JavaScript za aktiviranje uređaja za Internet Of Things (Internet „stvari“), koji se pokreću na malim mikroprocesorima ugrađenim u njih. **Espruino** je čip mikrokontrolera koji je namenski projektovan za pokretanje JavaScripta. Stoga, kad naučite JavaScript, moći ćete da gradite veb sajtove i aplikacije za mobilne uređaje i čak da kontrolišete mikroprocesore ugrađene u uređaje. JavaScript postaje sve popularniji, podržan je na sve više hardvera i prožima kroz skoro svaku oblast računarstva.

JavaScript jezik nije težak za učenje, ali predstavlja izazov kada pišemo velike, složene programe. Jedan od izazova ovog jezika je što je interpretirani jezik i stoga ne postoji korak kompajliranja. Da li ste napravili neku grešku u sintaksi znaćete samo kada pokrenete celu aplikaciju kroz interpreter tokom pokretanja. Još jedan izazov je što ovo nije objektno-orijentisani jezik i potrebne su velika pažnja i disciplina za izgradnju dobrog JavaScripta koji je jednostavan za održavanje i koji je razumljiv. Za programere koji su koristili druge objektno-orijentisane jezike, kao što su Java, C# ili C++, JavaScript može da izgleda kao potpuno strano okruženje.

TypeScript premošćava ovaj jaz. On je strogo tipiziran, objektno-orijentisan jezik koji koristi kompajler za generisanje JavaScripta. Stoga, omogućava da upotrebimo dobro poznate objektno-orijentisane tehnike i obrasce projektovanja za izgradnju JavaScript aplikacija. Imajte na umu da je JavaScript koji je generisan pomoću TypeScripta samo običan Java Script i stoga će se pokretati gde god može da se pokreće – u pretraživaču, na serveru, na mobilnom uređaju ili na ugrađenom uređaju.

Ovo poglavlje je podeljeno u dva glavna odeljka. Prvi odeljak sadrži pregled nekih prednosti upotrebe TypeScripta, a drugi je posvećen podešavanju TypeScript razvojnog okruženja.

Ako ste iskusni TypeScript programer i već imate podešeno radno okruženje, možete da preskočite ovo poglavlje. Ako nikada ranije niste koristili TypeScript, a kupili ste ovu knjigu zato što želite da razumete šta TypeScript može da radi, onda nastavite čitanje.

U ovom poglavlju obradićemo sledeće teme:

- prednosti TypeScripta
  - kompajliranje
  - strogo tipiziranje
  - integracija sa popularnim JavaScript bibliotekama
  - kapsuliranje
  - privatne i javne promenljive članovi
- podešavanje radnog okruženja
  - Visual Studio
  - WebStorm
  - Visual Studio Code
  - ostali editori i Grunt

UVOD U TYPESCRIPT

TypeScript je i jezik i set alatki za generisanje JavaScripta. Projektovao ga je Anders Hejlsberg iz „Microsofta“ (dizajner C#-a). TypeScript je projekat otvorenog koda koji pomaže programerima da pišu velike JavaScript projekte.

On generiše JavaScript. Umesto da zahteva potpuno novo radno okruženje, JavaScript generisan pomoću TypeScripta može ponovo da upotrebi sve postojeće alatke JavaScripta, radne okvire i biblioteke već dostupne za JavaScript. Međutim, TypeScript jezik i kompajler približavaju razvoj JavaScripta tradicionalnijem objektno-orijentisanom programiranju.

ECMAScript standard

JavaScript postoji veoma dugo i upravljan je standardom funkcija jezika. Jezik definisan u ovom standardu naziva se ECMAScript i svaki JavaScript interpreter mora da isporuči funkcije i elemente koji su u skladu sa ovim standardom. Definicija ovog standarda pomogla je rast JavaScripta i Veba uopšte i omogućava da se veb sajtovi pravilno renderuju na mnogim različitim pretraživačima na mnogo različitih operativnih sistema. ECMAScript standard, publikovan 1999. godine, poznat je kao **ECMA-262** (treće izdanje).

Zbog velike popularnosti jezika i eksplozivnog rasta internet aplikacija, bilo je potrebno da se izvrše revidiranje i ažuriranje ECMAScript standarda. Ovaj proces revizije rezultirao je ažuriranim nacrtom specifikacije za ECMAScript, pod nazivom četvrto izdanje. Nažalost, u tom nacrtu je takođe predlagana potpuna prepravka jezika i stoga nije dobro prihvaćen. Na kraju, rukovodioci iz „Yahooa“, „Googlea“ i „Microsofta“ pripremili su alternativni predlog, koji su nazvali **ECMAScript 3.1**. Ovaj predlog je numerisan kao 3.1, jer je to bio manji set funkcija trećeg izdanja, postavljen između trećeg i četvrtog izdanja standarda.

Predlog za potpunu prepravku jezika je na kraju prihvaćen kao peto izdanje standarda i nazvan je ECMAScript 5. ECMAScript četvrto izdanje nikada nije publikovano, ali je odlučeno da se uključe najbolje funkcije iz njega i izdanja 3.1 u šesto izdanje pod nazivom **ECMAScript Harmony**.

TypeScript kompajler ima parametar koji može da prebacuje kod između različitih verzija ECMAScript standarda. TypeScript trenutno podržava ECMAScript 3, ECMAScript 5, ECMAScript 6, pa čak i ECMAScript 7 (poznat i kao ECMAScript 2016).

Kada se kompajler pokrene za TypeScript, generisaće greške kompajlera ako kod koji pokušavate da kompajlirate nije validan za konkretni standard. Tim stručnjaka u „Microsoftu“ se obavezao da će pratiti sledeće ECMAScript standarde u svakoj novoj verziji Type Script kompajlera, pa će usvajanjem novih izdanja TypeScript jezik i kompajler pratiti ovaj standard.

Objašnjenje finijih detalja onoga što je uključeno u svako izdanje ECMAScript standarda nije preedviđeno u ovoj knjizi, ali je važno da znate da postoje razlike između verzija standarda. Neke verzije pretraživača ne podržavaju ES5 (IE8 je primer), ali većina ga podržava. Kada birate verziju ECMAScript za projekte, treba da razmotrite koje verzije pretraživača će biti podržane ili koji standard podržava JavaScript izvršavanje koda.

Prednosti TypeScripta

Da biste što bolje uočili neke od prednosti TypeScripta, pogledajte neke od mogućnosti koje donosi TypeScript:

- korak kompajliranja
- strogo ili statično tipiziranje
- definicije tipa za popularne JavaScript biblioteke
- kapsuliranje
- dekoratori privatnih i javnih promenljivih članova

Kompajliranje

Jedna od okolnosti koje najviše deluju frustrirajuće u vezi JavaScript razvoja je nedostatak koraka kompajliranja. JavaScript je interpretirani jezik i, stoga, treba da bude pokrenut nasuprot interpretera da bi bila testirana validnost koda. Svaki JavaScript programer ima „horor priče“ o satima provedenim u pokušajima da pronađe greške u kodu samo da bi otkrio da je izostavio običnu zatvorenu zagradu {, ili jednostavan zarez , ili je postavio dvostruki navodnik „ na mestu na kojem treba da stoji jednostruki navodnik ‚. Još je gora situacija kada se pogrešno napiše naziv svojstva ili se slučajno dodeli neka globalna promenljiva.

TypeScript će kompajlirati kod i generisaće greške kompajliranja na mestima gde pronađe ove vrste grešaka u sintaksi. To je očigledno veoma korisno i može pomoći u isticanju grešaka pre nego što je JavaScript i pokrenut. U velikim projektima programeri često treba da izvrše velika spajanja koda –kada se koriste popularne alatke koje spajanje izvršavaju automatski, iznenađujuće je koliko često kompajler otkrije greške.

Iako alatke koje vrše ovu vrstu provere sintakse, kao što je JSLint, postoje već godinama, očigledno je velika prednost ako su ove alatke integrisane u setu razvojnih alata. Upotreba TypeScript kompajlera u okruženju neprekidne integracije takođe će biti neuspešna kada se pronađu greške kompajliranja, dalje štiteći osnovu koda od ovih vrsta grešaka.

Strogo tipiziranje

JavaScript nije strogo tipiziran. To je jezik koji je veoma dinamičan i, stoga, omogućava objektima da menjaju svojstva i ponašanje u toku rada. Kao primer, pogledajte sledeći kod:

```
var test = "this is a string";
test = 1;
test = function(a, b) {
    return a + b;
}
```

U prvoj liniji isečka koda promenljiva `test` je povezana sa znakovnim nizom. Zatim je dodeljena broju i na kraju je ponovo definisana da bi postala funkcija koja očekuje dva parametra. To znači da je tip promenljive `test` promenjen od znakovnog niza do broja, a zatim je ova promenljiva postala funkcija. Međutim, tradicionalni objektno-orijentisani jezici neće omogućiti promenu tipa promenljive i zbog toga se nazivaju strogo tipizirani jezici.

Iako je ceo prethodni kod validan JavaScript, pa, stoga, može biti opravdan, prilično je lako uočiti kako on može da izazove grešku u vreme pokretanja prilikom izvršenja. Zamislite da ste odgovorni za pisanje funkcije biblioteke za dodavanje dva broja, a da je neki drugi programer slučajno ponovo dodelio funkciju za oduzimanje ovih brojeva.

Ove vrste grešaka mogu lako da se pronađu u nekoliko linija koda, ali je veoma teško pronaći ih i ispraviti kada se osnova koda i razvojni tim proširuju.

Još jedna funkcija strogog tipiziranja je da IDE u kojem radite „razume“ koji tip promenljive koristite i može da ponudi bolje opcije za automatsko završavanje ili Intellisense opcije.

„Sintaktički šećer" TypeScripta

TypeScript predstavlja veoma jednostavnu sintaksu za proveru tipa objekta u vreme kompajliranja. Ova sintaksa se naziva „sintaktički šećer", ili formalno, oznaka tipa. Pogledajte sledeći TypeScript kod:

```
var test: string = "this is a string";
test = 1;
test = function(a, b) { return a + b; }
```

Vidite da smo u prvoj liniji isečka koda uneli dvotačku (:) i ključnu reč `string` između promenljive i njene dodele. Ova sintaksa oznake tipa znači da podešavamo tip promenljive `test` na tip `string` i svaki kod koji ne tretira promenljivu `test` kao `string` će generisati grešku kompajliranja. Pokretanje prethodnog koda u TypeScript kompajleru generisaće dve greške:

```
hello.ts(3,1): error TS2322: Type 'number' is not assignable to
type
'string'.
hello.ts(4,1): error TS2322: Type '(a: any, b:
any) => any' is not assignable
to type 'string'.
```

Prva greška je prilično očigledna. Specifikovali smo da je promenljiva `test` tipa `string` i stoga će pokušaj dodele broja ovoj promenljivoj generisati grešku kompajliranja. Druga greška je slična prvoj i u suštini ukazuje da ne možemo da dodelimo funkciju u znakovni niz.

TypeScript kompajler predstavlja strogo ili statično tipiziranje u JavaScript kodu, pružajući prednosti strogo tipiziranog jezika. TypeScript je, stoga, opisan kao nadskup JavaScripta. Ovo ćemo detaljnije istražiti u sledećem poglavlju.

Definicije JavaScripta i TypeScripta

Kao što ste videli, TypeScript ima mogućnost da označi JavaScript i uvodi strogo tipiziranje u razvoj JavaScripta. Međutim, kako da izvršimo strogo tipiziranje u postojećim JavaScript bibliotekama? Drugim rečima, ako imamo postojeću JavaScript biblioteku, kako da je integrišemo za upotrebu unutar TypeScripta? Odgovor je iznenađujuće jednostavan – kreiranjem fajla definicije. TypeScript koristi fajlove sa ekstenzijom .d.`ts` kao vrstu fajla zaglavlja, slično jezicima kao što je C++, za nametanje strogog tipiziranja u postojeće JavaScript biblioteke. Ovi fajlovi definicije sadrže informacije koje opisuju svaku dostupnu funkciju i/ili promenljivu, zajedno sa njihovim povezanim oznakama tipa.

Pogledajte kako izgleda definicija. Kao primer, pogledajte JavaScript funkciju `describe` iz popularnog radnog okvira za testiranje koda Jasmine:

```
var describe = function(description, specDefinitions) {
  return jasmine.getEnv().describe(description,
  specDefinitions);
};
```

Vidite da ova funkcija ima dva parametra - `description` i `specDefinitions`. Nažalost, JavaScript nam ne ukazuje kog su tipa ove promenljive. Potrebno je da pregledamo Jasmine dokumentaciju da bismo otkrili kako da pozovemo ovu funkciju i koje su promenljive očekivane za oba parametra. Na stranici http://jasmine.github.io/2.0/introduction.html videćete primer kako se upotrebljava ova funkcija:

```
describe("A suite", function () {
  it("contains spec with an expectation", function () {
    expect(true).toBe(true);
  });
});
```

Iz dokumentacije možete lako da vidite da je prvi parametar `string`, a drugi function. Međutim, ne postoji ništa u JavaScriptu što nas primorava da kod uskladimo sa ovom API definicijom. Kao što sam ranije pomenuo, može lako da se desi da nepravilno pozovemo ovu funkciju, koristeći, na primer, dva broja, ili da prvo pošaljemo funkciju, pa znakovni niz. Ovakve greške će očigledno generisati greške prilikom pokretanja koda. Međutim, upotreba jednostavnog TypeScript fajla definicije će generisati greške u vreme kompajliranja, čak i pre nego što pokušamo da pokrenemo ovaj kod.

Pogledajte sada odgovarajuću TypeScript definiciju za ovu funkciju, koja se nalazi u fajlu definicije `jasmine.d.ts`:

```
declare function describe(
  description: string,
  specDefinitions: () => void
): void;
```

Ovde imamo TypeScript definiciju za Jasmine funkciju `describe`. Ona izgleda veoma slično samoj funkciji, ali daje malo više informacija o parametrima.

Jasno je da je parametar `description` strogo tipiziran na tip `string`, a parametar `specDefinitions` je strogo tipiziran na tip function, koji vraća vrednost void. TypeScript koristi sintaksu dvostruke zagrade `()` za deklarisanje funkcija, a sintaksu strelice za prikaz vraćenog tipa funkcije. Dakle, `() =>` void je funkcija koja ne vraća vrednost. Na kraju, opis same funkcije će, takođe, vratiti vrednost void.

Ako naš kod treba da prođe u funkciju kao prvi parametar, a znakovni niz kao drugi parametar (jasno kršeći definiciju ove funkcije) na sledeći način:

```
describe(() => { /* function body */}, "description");
```

TypeScript će generisati sledeću grešku:

```
hello.ts(11,11): error  TS2345: Argument of type  '()
=> void'  is not assignable to parameter of type
'string'.
```

Ova greška nam ukazuje da smo pokušali da pozovemo funkciju `describe` korišćenjem nevalidnih parametara. Predstavićemo detaljnije fajlove definicije u sledećim poglavljima, ali ovaj primer jasno pokazuje da će TypeScript kompajler generisati greške ako pokušamo nepravilno da upotrebimo eksterne JavaScript biblioteke.

DefinitelyTyped

Ubrzo nakon što je izdat TypeScript, Boris Yankov je pokrenuo GitHub skladište za čuvanje fajlova definicije, pod nazivom DefinitelyTyped (http://definitelytyped.org). Ovo skladište je sada postala prva stanica poziva za integrisanje eksternih JavaScript biblioteka u TypeScript i trenutno skladišti definicije za više od 1.600 JavaScript biblioteka.

Kapsuliranje

Jedan od osnovnih principa objektno-orijentisanog programiranja je kapsuliranje – mogućnost definisanja podataka, kao i seta funkcija koje mogu da se izvršavaju na konkretnim podacima u jednoj komponenti. Većina programskih jezika ima koncept klase za ovu namenu, obezbeđujući način za definisanje šablona za podatke i povezane funkcije.

Pogledajte prvo jednostavnu definiciju TypeScript klase:

```
class MyClass {
  add(x, y) {
     return x + y;
  }
}

var classInstance = new MyClass();
var result = classInstance.add(1,2);
console.log(`add(1,2) returns ${result}`);
```

Ovaj kod je prilično jednostavan za čitanje i razumevanje. Kreirali smo klasu i dodelili joj naziv `MyClass` jednostavnom funkcijom `add`. Da bismo upotrebili ovu klasu, jednostavno ćemo kreirati njenu instancu i pozvati funkciju `add`, koristeći dva argumenta.

Nažalost, JavaScript nema iskaz klase, pa za reprodukovanje funkcionalnosti klasa koristi funkcije. Kapsuliranje kroz klase se izvršava upotrebom obrasca prototipa ili upotrebom obrasca zatvorenog izraza. Razumevanje obrazaca prototipova i zatvorenih izraza i njihova pravilna upotreba smatraju se osnovom kada pišete velike JavaScript projekte.

Zatvoreni izraz je, u stvari, funkcija koja se odnosi na nezavisne promenljive, što znači da promenljive koje su definisane unutar funkcije zatvorenog izraza „pamte“ okruženje u kojem su kreirane. To obezbeđuje JavaScriptu način za definisanje lokalnih promenljivih i obezbeđuje kapsuliranje. Definicija klase `MyClass` u prethodnom kodu upotrebom zatvorenog izraza u JavaScriptu izgledaće slično sledećem kodu:

```
var MyClass = (function () {
  // the self-invoking function is the
  // environment that will be remembered
  // by the closure function MyClass() {
  // MyClass is the inner function,
  // the closure
  }
  MyClass.prototype.add = function (x, y) {
    return x + y;
  };
  return MyClass;
}());
var classInstance = new MyClass();
var result = classInstance.add(1, 2);
console.log("add(1,2) returns " + result);
```

Kod smo započeli promenljivom pod nazivom `MyClass`, koju smo dodelili funkciji koja je odmah izvršena – vidite sintaksu `})()`; blizu kraja definicije zatvorenog izraza. Ova sintaksa je napisana na uobičajeni način za JavaScript da bi se izbegao prelazak promenljivih u globalni imenski prostor. Zatim smo definisali novu funkciju pod nazivom `MyClass` i vratili je u spoljašnji poziv funkcije. Za ubacivanje druge funkcije u definiciju klase `MyClass` upotrebili smo ključnu reč `prototype`. Ova funkcija, pod nazivom `add`, koristi dva parametra i vraća vrednost njihovog zbira.

Poslednjih nekoliko linija koda prikazuju kako da upotrebimo ovaj zatvoreni izraz u Java Scriptu. Kreiraćemo instancu tipa zatvorenog izraza, a zatim ćemo izvršiti funkciju `add`. Kada se pokrene ovaj kod, biće evidentirano u konzoli da funkcija `add`(1,2) vraća vrednost 3, kao što se i očekuje.

Ako pogledate JavaScript kod nasuprot TypeScript koda, lako možete da uočite kako TypeScript kod izgleda jednostavno u poređenju sa ekvivalentnim JavaScript kodom. Sećate li se da smo napomenuli da JavaScript programeri mogu lako da zaborave veliku ili malu zagradu? Pogledajte poslednju liniju u definiciji zatvorenog izraza - `})();`. Ako je pogrešno uneta velika ili mala zagrada, to može da dovede do toga da satima tražite grešku koju treba ispraviti.

TypeScript klase generišu zatvorene izraze

JavaScript, kao što je prikazano u prethodnom primeru, je, u stvari, ispis definicije Type Script klase. Dakle, TypeScript generiše, umesto vas, zatvoreni izraz.

Godinama se već govori o dodavanju koncepta klasa u JavaScript jezik i on je trenutno deo ECMAScript šestog izdanja (Harmony) standarda. „Microsoft" se obavezao da će pratiti ECMAScript standard u TypeScript kompajleru kada ovi standardi budu publikovani.

Javni i privatni pristupni metod

Sledeći princip objektno-orijentisanog programiranja koji se koristi u kapsuliranju je koncept skrivanja podataka – mogućnost definisanja javnih i privatnih promenljivih. Privatne promenljive su skrivene od korisnika određene klase, jer treba da ih upotrebljava jedino sama klasa. Nenamerno otkrivanje ovih promenljivih može lako da izazove greške prilikom pokretanja koda.

Nažalost, JavaScript nema izvorni način da se promenljive deklarišu kao privatne. Iako ova funkcionalnost može da bude simulirana pomoću zatvorenih izraza, mnogi JavaScript programeri jednostavno koriste karakter donje crtice _ za označavanje privatne promenljive. Privatnoj promenljivoj, ako joj znate naziv, možete lako da dodelite vrednost prilikom izvršenja koda. Pogledajte sledeći JavaScript kod:

```
var MyClass = (function() {
     function MyClass() {
         this._count = 0;
     }
     MyClass.prototype.countUp = function() {
       this._count ++;
     }
     MyClass.prototype.getCountUp = function() {
       return this._count;
```

```
    }
    return MyClass;
}());

var test = new MyClass();
test._count = 17;
console.log("countUp : " + test.getCountUp());
```

Promenljiva `MyClass` je, u stvari, zatvoreni izraz sa funkcijom konstruktora, funkcijom `countUp` i funkcijom `getCountUp`. Promenljiva `_count` treba da bude promenljiva privatni član koja se koristi samo unutar oblasti važenja zatvorenog izraza. Konvencija imenovanja upotrebom donje crtice korisniku ove klase daje neki pokazatelj da je promenljiva privatna, ali će JavaScript i dalje omogućiti da manipulišete promenljivom `_count`. Pogledajte pretposlednju liniju isečka koda. Eksplicitno smo podesili vrednost promenljive `_count` na 17, što JavaScript dozvoljava, ali to nije poželjno za originalnog kreatora klase. Ispis ovog koda će biti: `countUp : 17`.

Međutim, TypeScript predstavlja ključne reči public i private (između ostalih), koje mogu da se upotrebe u promenljivim članovima klase. Pokušaj da pristupite promenljivoj članu klase koja je označena kao private generisaće grešku u vreme kompajliranja. Kao primer ovoga, prethodni JavaScript kod može da bude napisan u TypeScriptu na sledeći način:

```
class CountClass {
  private _count: number;
    constructor() {
      this._count = 0;
    }
  countUp() {
    this._count ++;
  }
  getCount() {
    return this._count;
  }
}
var countInstance = new CountClass() ;
countInstance._count = 17;
```

U drugoj liniji isečka koda deklarisali smo private promenljivu član pod nazivom `_count`. U kodu sada imamo konstruktor, funkciju `countUp` i funkciju `getCount`. Ako kompajliramo ovaj fajl, kompajler će generisati sledeću grešku:

```
hello.ts(39,15): error  TS2341: Property '_count' is  private and  only
accessible within class  'CountClass'
```

Ova greška je generisana zato što pokušavamo da pristupimo privatnoj promenljivoj `_count` u poslednjoj liniji koda.

TypeScript kompajler nam, stoga, pomaže da poštujemo javne i privatne metode pristupa generisanjem greške kompajlera kada nenamerno prekršimo pravilo pristupa.

Ne zaboravite - ovi pristupni metodi su funkcije samo u vreme kompajliranja i neće uticati na generisani JavaScript. Treba da imate ovo na umu ako pišete JavaScript biblioteke koje će koristiti nezavisni korisnici. Zapamtite da će, prema standardnom podešavanju, TypeScript kompajler i dalje generisati JavaScript izlazni fajl, čak i ako postoji greška prilikom kompajliranja. Ova opcija može da bude modifikovana da bi se nametnulo TypeScript kompajleru da ne generiše JavaScript ako postoje greške prilikom kompajliranja.

TYPESCRIPT IDE-OVI

Ovaj odeljak treba da vam pomogne da obezbedite spremno TypeScript okruženje da biste mogli da editujete, kompajlirate i pokrećete TypeScript kod i da ispravljate greške u njemu. TypeScript je izdat kao projekat otvorenog koda i uključuje varijantu za Windows i varijantu Node. To znači da će se kompajler pokretati na Windows, Linux, OS X i svakom drugom operativnom sistemu koji podržava Node. U Windows okruženju možemo da instaliramo Visual Studio, koji će registrovati `tsc.exe` (TypeScript kompajler) u direktorijum c:\Program Files, ili možemo da upotrebimo Node. Na Linux i OS X okruženjima potrebno je da upotrebimo Node.

U ovom odeljku pregledaćemo sledeće IDE-ove:

- kompajliranje zasnovano na Nodeu
- Visual Studio 2015
- WebStorm
- Visual Studio Code
- upotreba Grunta

Kompajliranje zasnovano na Nodeu

Najjednostavnije TypeScript razvojno okruženje sastoji se od jednostavnog editora za tekst i TypeScript kompajlera zasnovanog na Nodeu. Pogledajte Node veb sajt (https://nodejs.org/) i pratite uputstva za instaliranje Nodea na operativni sistem po vašem izboru.

Kada je Node instaliran, TypeScript može da bude instaliran jednostavnim unosom sledeće komande:

```
npm install –g typescript
```

Ova komanda poziva Node Package Manager (`npm`) da instalira TypeScript kao globalni modul (opcija -g), koji će biti dostupan bez obzira u kojem se direktorijumu trenutno nalazite. Kada je TypeScript instaliran, možete da prikažete aktuelnu verziju kompajlera unosom sledeće komande:

```
tsc –v
```

U vreme pisanja ove knjige aktuelna verzija TypeScript kompajlera je version 2.1.5 i stoga će ispis ove komande biti sledeći:

```
Version 2.1.5
```

Kreirajte sada TypeScript fajl pod nazivom hello.`ts`, koristeći sledeći sadržaj:

```
console.log('hello  TypeScript');
```

Iz komandne linije možemo da upotrebimo TypeScript da bismo kompajlirali ovaj fajl u JavaScript fajl upotrebom komande:

```
tsc hello.ts
```

Kada je TypeScript završio zadatak, generisaće fajl `hello.js` u aktuelnom direktorijumu.

Kreiranje fajla tsconfig.json

TypeScript kompajler koristi fajl `tsconfig.json` u osnovnom direktorijumu projekta za specifikovanje svih globalnih podešavanja TypeScript projekta i opcija kompajlera. To znači da, umesto da kompajliramo TypeScript fajlove, jedan po jedan (specifikovanjem svakog fajla u komandnoj liniji), možemo jednostavno da unesemo komandu `tsc` u osnovni direktorijum projekta, a TypeScript će rekurzivno pronaći i kompajlirati sve TypeScript fajlove unutar osnovnog direktorijuma i svih poddirektorijuma. Fajl `tsconfig.json` koji je potreban TypeScriptu da bi izvršio ovaj zadatak može da bude kreiran iz komandne linije jednostavnim unosom sledeće komande:

```
tsc --init
```

Rezultat ove komande je osnovni `tsconfig.json` fajl, kao što je ovde prikazano:

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "noImplicitAny": false,
    "sourceMap": false
   }
}
```

Ovo je jednostavan fajl JSON formata, sa jednim JSON svojstvom, pod nazivom `compiler Options`, koje specifikuje opcije kompajlera za projekat. Svojstvo target ukazuje na željeni JavaScript izlaz za generisanje i može da bude `es3`, `es5`, `es6`, `ES2016`, `ES2017` ili `ESNext`. Opcija pod nazivom `sourceMap` je oznaka koja ukazuje da li treba generisati izvorne mape koje se koriste za ispravljanje grešaka. Opcija `noImplicitAny` je oznaka koja ukazuje da moramo da pokušamo da strogo tipiziramo sve promenljive pre upotrebe.

TypeScript omogućava postojanje više tsconfig.json fajlova unutar strukture direktorijuma, tako da različiti poddirektorijumi mogu da upotrebe različite opcije kompajlera.

Kada je fajl `tsconfig.json` kreiran, možemo da kompajliramo aplikaciju, tako što ćemo jednostavno uneti sledeću komandu:

```
tsc
```

Ova komanda će pozvati TypeScript kompajler, koristeći fajl `tsconfig.json` koji smo kreirali za generisanje `hello.js` JavaScript fajla. U stvari, svaki izvorni fajl TypeScripta koji ima ekstenziju `.ts` će generisati JavaScript fajl sa ekstenzijom .js. Sada možemo da pokrenemo aplikaciju, tako što ćemo uneti sledeći kod:

```
node  hello.js
```

Pošto aplikacija jednostavno evidentira neki tekst u komandnoj liniji, ispis će biti sledeći:

```
λ node  hello.js
hello  TypeScript
```

Jednostavnim editorom za tekst i pristupom komandnoj liniji uspešno smo kreirali jednostavno TypeScript radno okruženje koje je zasnovano na Nodeu.

Microsoft Visual Studio

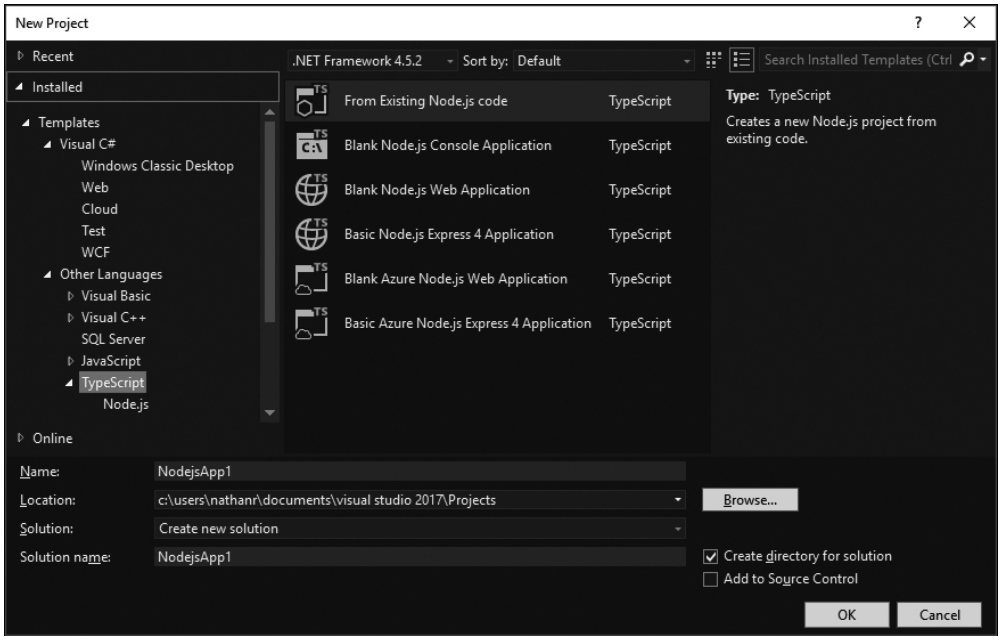
Pogledajte sada „Microsoftov“ Visual Studio. Ovo je „Microsoftov“ primarni IDE. Postoje različite kombinacije u njegovoj ceni. U vreme pisanja ove knjige „Microsoft“ je izdao Visual Studio 2017 Release Candidate, kao naslednika verzije Visual Studio 2015. „Microsoft“ ima model licenciranja zasnovan na Azureu, sa početnom cenom od oko 45 dolara mesečno, pa sve do profesionalne licence sa MSDN pretplatom od oko 1.199 dolara. Dobra vest je da „Microsoft“ ima i Community izdanje, koje može da se koristi u privatnim okruženjima besplatno. TypeScript kompajler je uključen u sva ova izdanja.

Visual Studio može da se preuzme kao veb instaler ili kao .iso slika diska. Imajte na umu da će veb instaler zahtevati internet konekciju u toku procesa instalacije, jer preuzima potrebne pakete u toku koraka instalacije. Visual Studio će takođe zahtevati Internet Explorer 10 ili noviji, ali će vas upozoriti na to u toku instalacije ako još niste nadgradili pretraživač. Ako koristite .iso instaler, imajte na umu da će možda biti potrebno da preuzmete i instalirate dodatne „zakrpe“ operativnog sistema, ako ga niste skoro ažurirali.

Kreiranje Visual Studio projekta

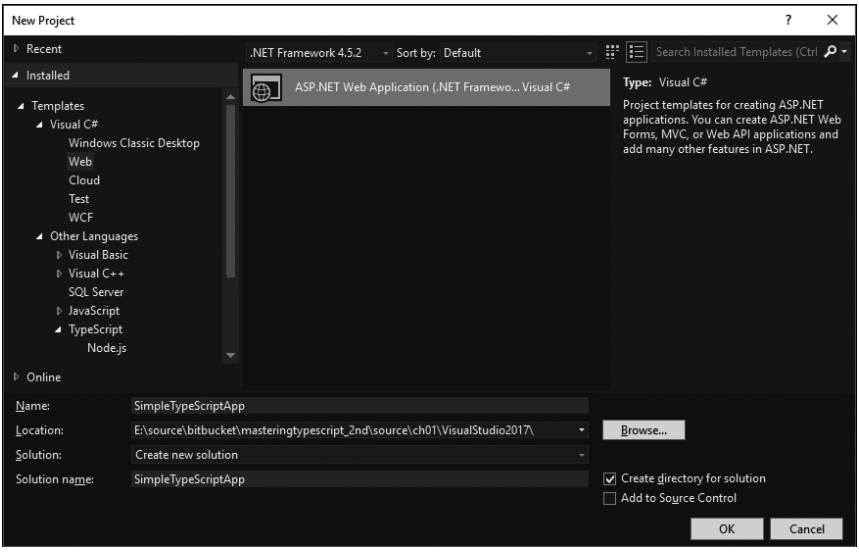
Kada je Visual Studio 2017 instaliran, pokrenite ga i kreirajte novi projekat (**File | New Project**). Postoje mnoge različite opcije koje su dostupne za nove šablone projekta, u zavisnosti od izbora jezika. U odeljku **Templates** sa leve strane videćete opciju **Other Languages** i pod njom opciju **TypeScript**. Šabloni projekta koji su dostupni malo su drugačiji u verziji Visual Studio 2017 od onih iz verzije Visual Studio 2015, a usmereni su ka Node razvoju.

Visual Studio 2015 ima šablon pod nazivom Html Application with TypeScript, koji će kreirati veoma jednostavnu jednostranu veb aplikaciju. Nažalost, ova opcija je uklonjena iz verzije Visual Studio 2017, kao što je prikazano na sledećoj slici:



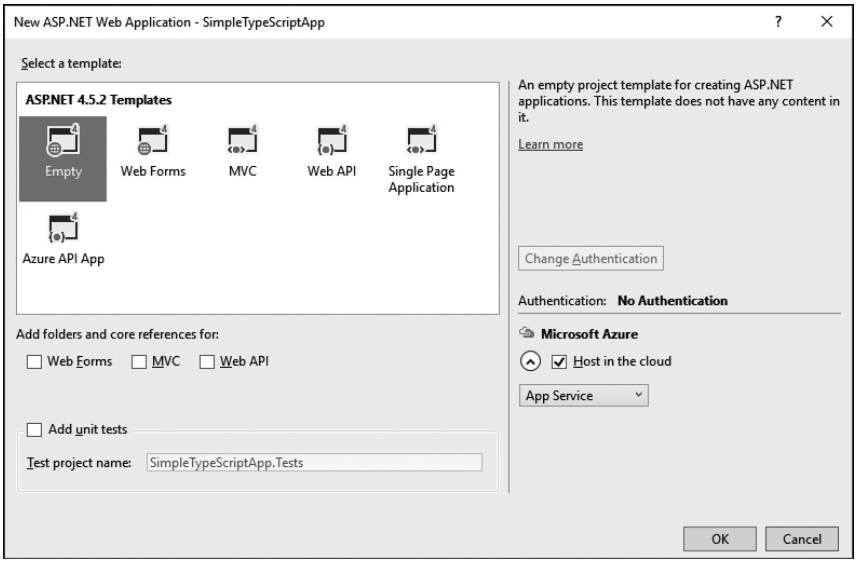
Visual Studio 2017 – šabloni TypeScript projekta

Da bismo kreirali jednostavnu TypeScript veb aplikaciju u Visual Studiou 2017, potrebno je da kreiramo prvo praznu veb aplikaciju, a zatim možemo da dodamo TypeScript fajlove u ovaj projekat po potrebi. Iz okvira za dijalog **Templates** selektovaćemo opciju šablona **Visual C#** i selektovaćemo opciju Web. Otvoriće se šablon projekta pod nazivom **ASP.NET Web Application**. Selektovaćemo naziv (**Name**) i lokaciju (**Location**) za novi projekat, a zatim ćemo kliknuti na **OK**, kao što je prikazano na sledećoj slici:



Visual Studio 2017 – kreiranje ASP.NET veb aplikacije

Kada smo selektovali osnovne informacije za novi projekat, Visual Studio će generisati drugi okvir za dijalog sa pitanjem koju vrstu ASP.NET projekta bismo želeli da generišemo. Selektovaćemo šablon Empty, a zatim ćemo kliknuti na OK, kao što je prikazano na slici:

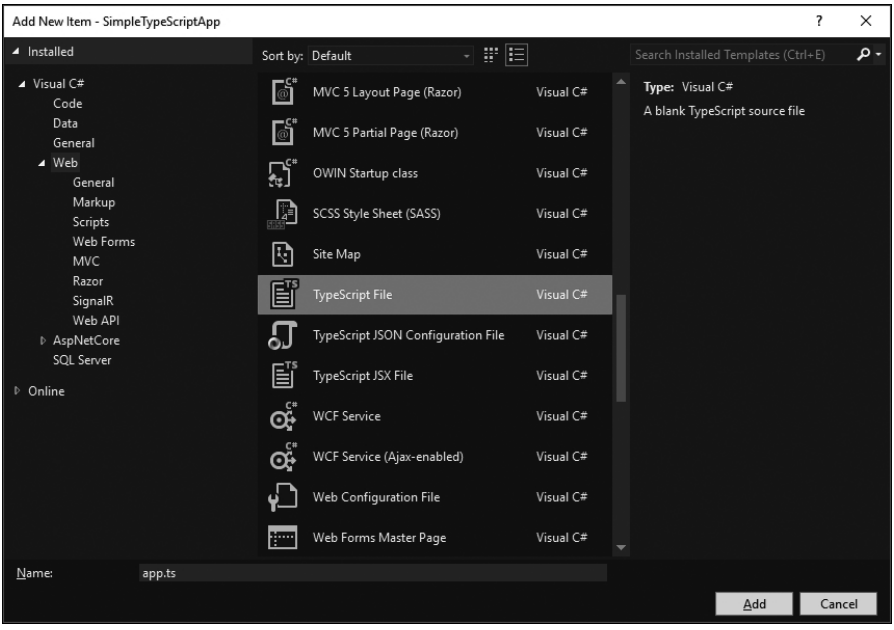


Visual Studio 2017 – opcije za kreiranje ASP.NET veb aplikacije

Visual Studio 2017 će, zatim, otvoriti sledeći okvir za dijalog, pod nazivom **Create App Service**, koji obezbeđuje opcije za kreiranje hosta u Azureu za novu veb aplikaciju. Mi nećemo publikovati aplikaciju na Azure, pa možemo da kliknemo na dugme **Skip** u ovom okviru za dijalog.

Standardna podešavanja projekta

Kada je kreirana nova Empty ASP.NET veb aplikacija, možemo da započnemo dodavanje fajlova u projekat, tako što ćemo kliknuti desnim tasterom miša na sam projekat i izabrati opciju **Add**, pa opciju **New Item**. Postoje dva fajla koja ćemo dodati u projekat: `index.html` i `app.ts` TypeScript. Za svaki od ovih fajlova selektovaćemo odgovarajući Visual Studio šablon na sledeći način:



Visual Studio – dodavanje TypeScript fajla

Sada možemo da otvorimo fajl `app.ts` i započnemo kucanje sledećeg koda:

```
class MyClass {
  public render(divId: string, text: string) {
     var el: HTMLElement = document.getElementById(divId);
     el.innerText = text;
  }
}
```

```
window.onload = () => {
  var myClass = new MyClass();
  myClass.render("content", "Hello World");
};
```

Ovde smo kreirali klasu pod nazivom `MyClass`, koja ima jednu funkciju `render`. Ova funkcija koristi dva parametra: `divId` i `text`. Funkcija pronalazi element HTML DOM koji se podudara sa argumentom `divId` i podešava svojstvo `innerText` na vrednost argumenta `text`. Zatim ćemo definisati funkciju koja će biti pozvana kada pretraživač pozove `window.onload`. Ova funkcija kreira novu instancu klase `MyClass` i poziva funkciju `render`.

Nemojte se uzbuđivati ako su ova sintaksa ili kod malo zbunjujući. Opisaćemo sve elemente jezika i sintaksu u narednim poglavljima. Poenta ove vežbe je jednostavno da upotrebimo Visual Studio kao razvojno okruženje za editovanje TypeScript koda.

Videćete da Visual Studio ima veoma moćne Intellisense opcije i predlaže kod, nazive funkcije ili nazive promenljive dok pišete kod. Ako predlozi ne budu prikazani automatski, pritisnite Ctrl–razmaknicu da biste prikazali Intellisense opcije za kod koji trenutno kucate.

Kada je postavljen fajl `app.ts`, možete da ga kompajlirate, tako što ćete pritisnuti tastere *Ctrl–Shift–B* ili *F6*, ili selektovati opciju **Build** iz linije sa alatkama. Ako postoje greške u TypeScript kodu koji kompajlirate, Visual Studio će automatski prikazati **Error List** panel koji sadrži greške kompajliranja. Dvostruki klik na bilo koju od ovih grešaka će otvoriti fajl u panelu editora i automatski će postaviti kursor na pogrešan kod.

Generisani fajl app.js nije uključen u Solution Explorer u Visual Studiou. Uključen je samo app.ts TypeScript fajl. To je standardno podešavanje. Ako želite da vidite generisani JavaScript fajl, jednostavno kliknite na dugme Show All Files u liniji sa alatkama Solution Explorera.

Da bismo uključili TypeScript fajl u HTML stranicu, potrebno je da editujemo fajl `index.html` i da dodamo oznaku `<script>` za učitavanje fajla `app.js` na sledeći način:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title></title>
  <script src="app.js"></script>
</head>
<body>
```

```
    <div id="content"></div>
</body>
</html>
```

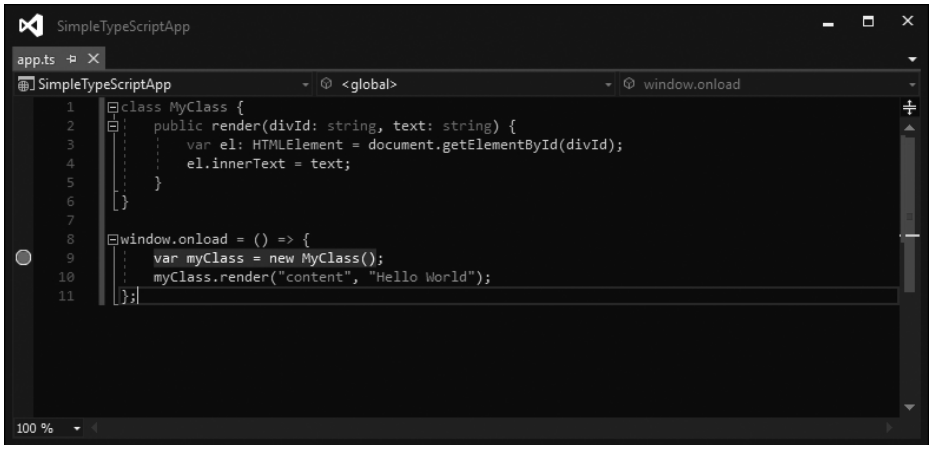
Ovde smo dodali oznaku `<script>` za učitavanje fajla `app.js` i kreirali smo element `<div>` sa id-om `content`. Ovo je DOM element čije će svojstvo modifikovati kod. Sada možemo da pritisnemo taster F5 da bismo pokrenuli aplikaciju:

Visual Studio index.html pokrenut u pretraživaču Chrome

Ispravljanje grešaka u Visual Studiou

Jedna od najboljih funkcija Visual Studioa je što je to integrisano okruženje. Ispravljanje grešaka TypeScripta u Visual Studiou je potpuno isto kao i ispravljanje grešaka jezika C# ili bilo kod drugog jezika u Visual Studiou, a uključuje uobičajene prozore Immediate, **Locals**, **Watch** i **Call stack**.

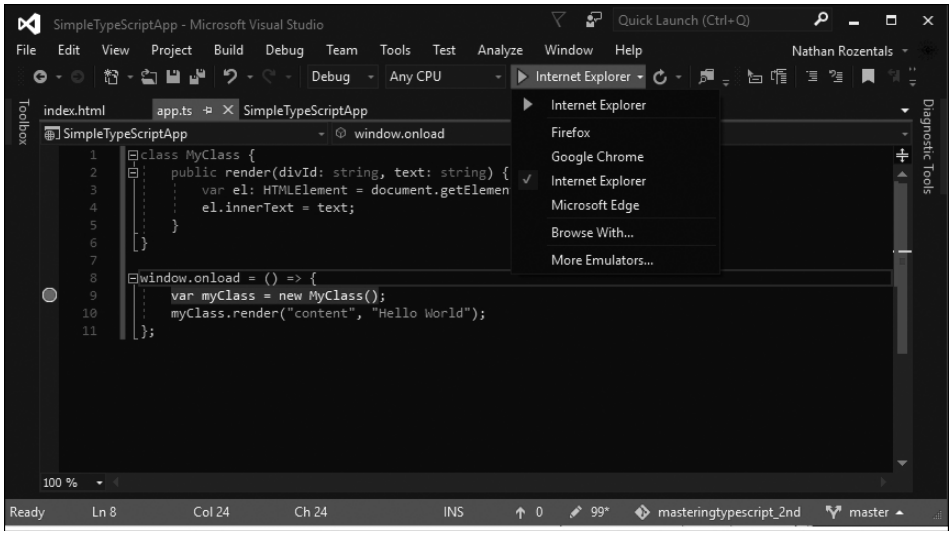
Da biste ispravili TypeScript u Visual Studiou, jednostavno postavite tačku prekida na liniju koju želite da prekinete u TypeScript fajlu (pomerite miš u područje tačke prekida pored linije izvornog koda i kliknite). Na sledećoj slici postavili smo tačku prekida unutar funkcije `window.onload`. Da biste počeli ispravljanje greške, potrebno je samo da pritisnete *F5*:



Visual Studio TypeScript editor sa postavljenom tačkom prekida u kodu

Kada je linija izvornog koda istaknuta žutom bojom, postavite kursor na bilo koju od promenljivih u kodu ili upotrebite prozore **Immediate**, **Watch**, **Locals** ili **Call stack**.

Imajte na umu da Visual Studio podržava ispravljanje grešaka samo u Internet Exploreru 11. Ako imate instalirano više pretraživača na mašini (uključujući i Microsoft Edge), uverite se da ste u liniji sa alatkama Debug izabrali Internet Explorer, kao što je prikazano na sledećoj slici:



Visual Studio linija sa alatkama Debug prikazuje opcije pretraživača.

WebStorm

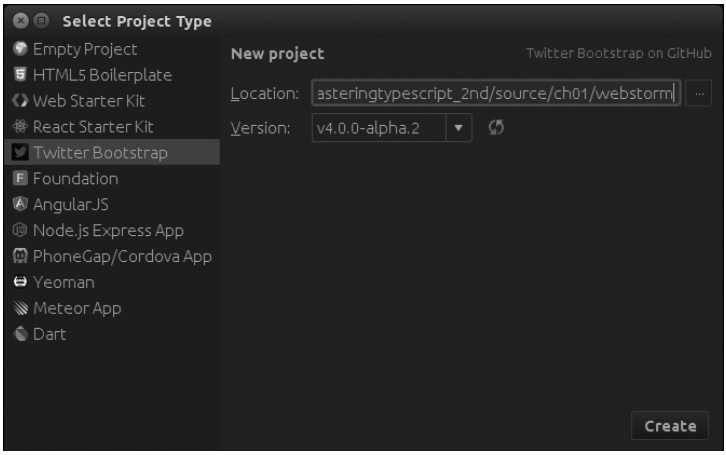
WebStorm je popularni IDE JetBrainsa (http://www.jetbrains.com/`webstorm`/), a pokreće se na Windows, Mac OS X i Linux sistemima. Cena se kreće od 59 dolara godišnje za jednog programera do 129 dolara godišnje za komercijalnu licencu. JetBrains nudi i probnu verziju od 30 dana.

WebStorm ima nekoliko odličnih funkcija, uključujući „živo" editovanje i predloge za kod ili Intellisense. „Živo" editovanje omogućava da zadržite otvoren pretraživač koji će se automatski ažurirati na osnovu promena u CSS-u, HTML-u i JavaScriptu dok kucate. Predlozi koda, koji su dostupni i u drugom popularnom JetBrains proizvodu ReSharper, označiće kod koji ste napisali i ukazaće na bolje načine njegovog implementiranja. I WebStorm ima veliki broj šablona projekata. Ovi šabloni će automatski preuzeti i uključiti relevantne JavaScript ili CSS fajlove, kao što su Twitter, Bootstrap ili HTML5 šablon.

Na Windows sistemu podešavanje WebStorma je jednostavno. Treba samo da preuzmete paket sa veb sajta i pokrenete program za instalaciju. Na Linux sistemu WebStorm je obezbeđen kao tar ball. Kada je raspakovan, instalirajte WebStorm pokretanjem skripta `webstorm`.sh u direktorijumu `bin`. Imajte na umu da na Linux sistemima pokrenuta verzija Java mora da bude instalirana pre nego što se instalacija nastavi.

Kreiranje WebStorm projekta

Da biste kreirali WebStorm projekat, pokrenite WebStorm i kliknite na **File | New Project**. Selektujte naziv, lokaciju i tip projekta. Za ovaj projekat smo selektovali Twitter Bootstrap:



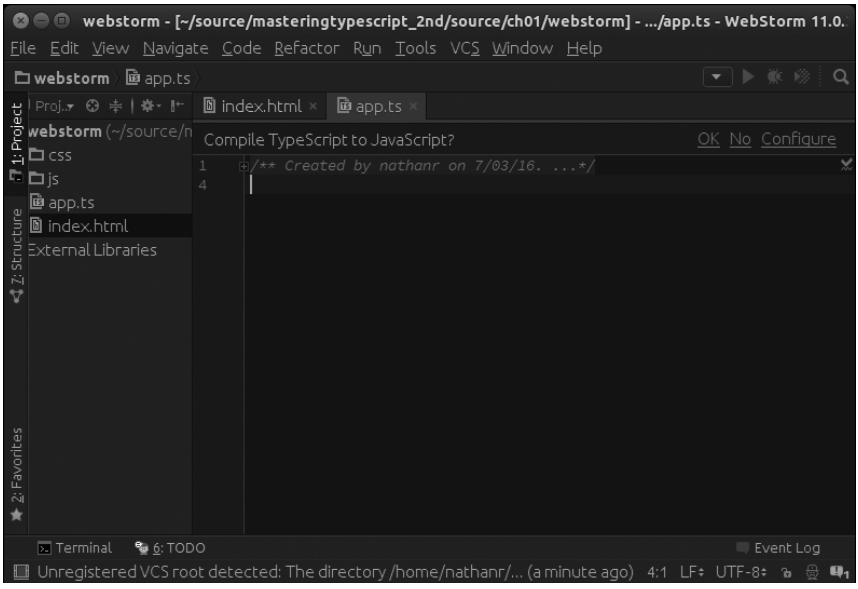
WebStorm okvir za dijalog za novi projekat

Standardni fajlovi

WebStorm je prikladno kreirao direktorijume css i js kao deo novog projekta i preuzeo je i uključio relevantne CSS i JavaScript fajlove da bismo mogli da započnemo izgradnju novog sajta zasnovanog na Bootstrapu. Vidite da WebStorm nije kreirao ni fajl `index.html`, ni TypeScript fajlove. Kreirajmo sada fajl `index.html`.

Jednostavno ćemo kliknuti na **File** | **New**, selektovaćemo HTML fajl i unećemo index kao naziv, a zatim ćemo kliknuti na **OK**.

Kreiraćemo TypeScript fajl na sličan način. Nazvaćemo ovaj fajl `app` (ili `app.ts`) da bismo reflektovali Visual Studio projekat. Kada kliknemo unutar novog `app.ts` fajla, WebStorm će prikazati poruku na vrhu fajla, koja uključuje predlog **Compile TypeScript to Java Script**? i tri opcije - **OK**, **No** i **Configure**, kao što je prikazano na sledećoj slici:



WebStorm edituje TypeScript fajl prvi put i prikazuje liniju za posmatranje fajla.

Klikom na **Configure** otvorićemo panel **Settings** zaTypeScript. Kliknite na polje za potvrđivanje **Enable TypeScript** compiler da biste omogućili modifikacije u podešavanjima, a zatim kliknite na radio-dugme **Use tsconfig.**`json` i na **OK**. WebStorm je sada konfigurisan da koristi fajl `tsconfig.json` u osnovnom direktorijumu projekta. Pošto ovaj fajl još ne postoji, otvoriće se panel za greške TypeScripta, ukazujući da kompajler ne može da pronađe fajl `tsconfig.json` u projektu. Da bismo ispravili ovu grešku, treba da kreiramo fajl `tsconfig.json`, pa ćemo, stoga, kliknuti na **File** | **New** i unećemo `tsconfig.json` kao naziv fajla. Vratićemo se na fajl `app.ts` i pritisnuti prečicu *Ctrl-S* da bismo snimili promene; poruka o grešci će nestati.

Izgradnja jednostavne HTML aplikacije

Sada, kada je WebStorm konfigurisan za kompajliranje TypeScript fajlova, kreirajmo jednostavnu TypeScript klasu i upotrebimo je za modifikovanje svojstva `innerText` HTML diva. Dok kucate, videćete WebStormovu funkciju za automatsko završavanje ili Intellisense funkciju, koja pomaže da upotrebite dostupne ključne reči, parametre, konvencije imenovanja i tako dalje. Ovo je jedna od najmoćnijih funkcija WebStorma i slična je poboljšanoj funkciji Intellisense iz Visual Studioa. Ukucajte sledeći TypeScript kod i videćete kako funkcioniše dostupna funkcija WebStorma za automatsko dovršavanje:

```
class MyClass {
  public render(divId: string, text: string) {
    var el: HTMLElement = document.getElementById(divId);
    el.innerText = text;
  }
}

window.onload = () => {
  var myClass = new MyClass();
  myClass.render("content", "Hello World");
}
```

Ovo je isti kod koji smo upotrebili u primeru za Visual Studio.

Ako imate greške u TypeScript fajlu, one će automatski biti prikazane u izlaznom prozoru, dajući vam instantnu povratnu informaciju dok kucate. Ovaj TypeScript fajl koji smo kreirali možemo da uključimo u `index.html` fajl i pokušamo da ispravimo greške.

Otvorite fajl `index.html` i dodajte oznaku `script` da biste uključili `app.js` JavaScript fajl, zajedno sa elementom `div` koji ima id `"content"`. Otkrićete da WebStorm ima moćne Intellisense funkcije kada editujete HTML:

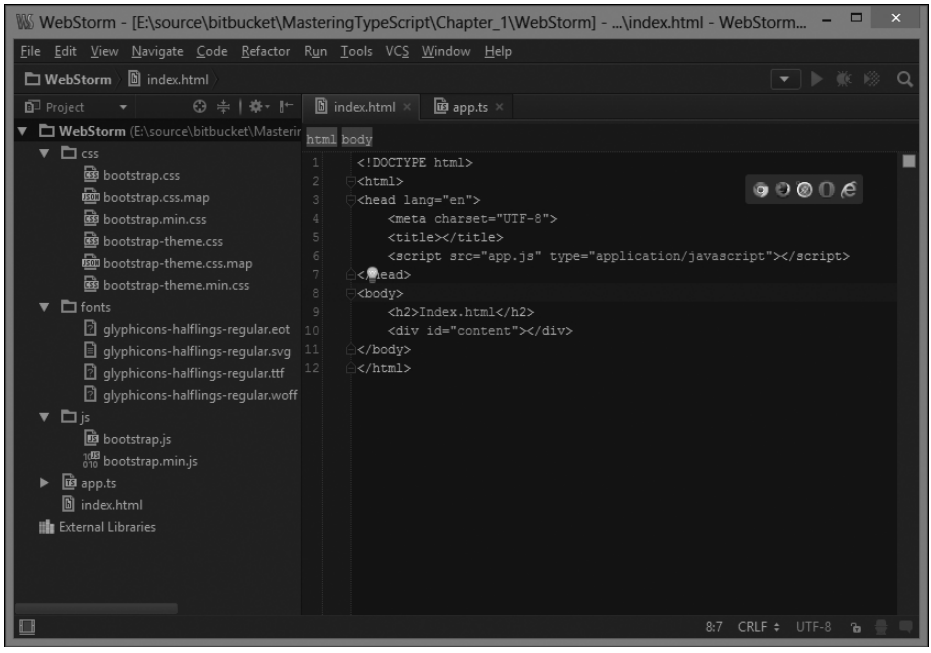
```
<!DOCTYPE html>
<html>
<head lang="en">
```

```
<meta charset="UTF-8">
<title></title>
<script src="app.js"></script>
</head>
<body>
  <div id="content"></div>
</body>
</html>
```

Ovaj HTML je isti kao onaj koji smo ranije upotrebili u primeru Visual Studioa.

Pokretanje veb stranice u Chromeu

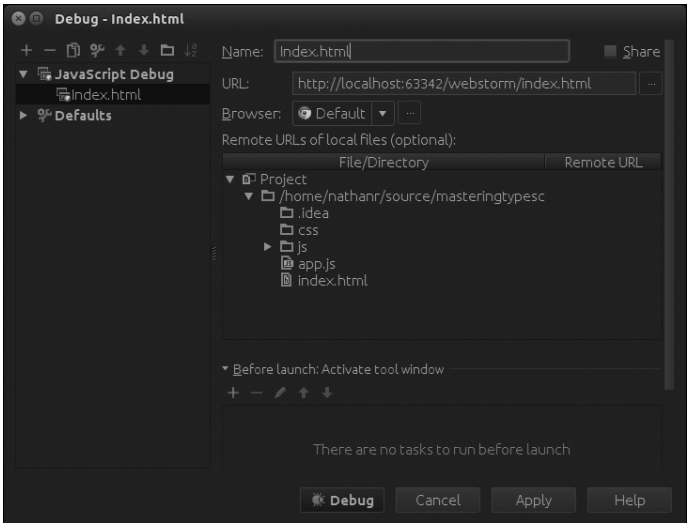
Kada pregledate ili editujete HTML fajlove u WebStormu, videćete mali set ikonica pretraživača koji se prikazuje u gornjem desnom uglu prozora editora. Kliknite na bilo koju ikonicu da biste otvorili aktuelnu HTML stranicu, koristeći izabrani pretraživač:



WebStorm editovanje HTML fajla prikazuje ikonice za pokretanje pretraživača.

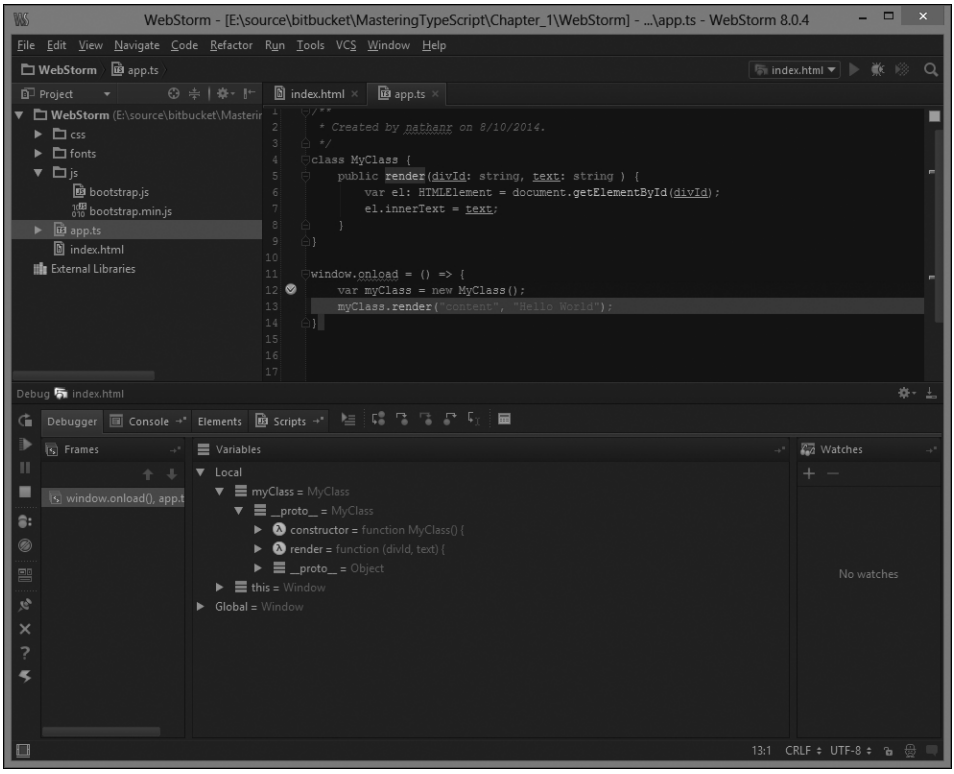
Ispravljanje grešaka u Chromeu

Da biste ispravili greške u veb aplikaciji pomoću WebStorma, potrebno je da podesite konfiguraciju za ispravljanje grešaka za fajl `index.html`. Kliknite na **Run** | **Debug**, pa editujte konfiguraciju. Kliknite na dugme plus (+), selektujte opciju JavaScript debug sa leve strane prozora i dodelite naziv konfiguraciji. Imajte na umu da je WebStorm već identifikovao da je `index.html` standardna stranica, ali to možete jednostavno da modifikujete. Zatim, kliknite na **Debug** na dnu ekrana, kao što je prikazano na sledećoj slici:



WebStorm konfiguracija za ispravljanje grešaka za `index.html`

WebStorm koristi dodatni modul Chromea da omogući ispravljanje grešaka u Chrome pretraživaču i kada prvi put započnete ispravljanje grešaka upozoriće vas da preuzmete i uključite JetBrains IDE Support Chrome dodatni modul. Kada je ovaj dodatni modul uključen, WebStorm ima veoma moćan set alata za istraživanje JavaScript koda, za posmatrače i za prikaz konzole i još mnogo korisnih alata unutar IDE-a.



WebStorm sesija za ispravljanje grešaka prikazuje panel za otkrivanje grešaka u kodu.

Visual Studio Code

Visual Studio Code, malo razvojno okruženje proizvedeno u „Microsoftu“, pokreće se na Windows, Linux i Mac sistemima. Uključuje razvojne funkcije, kao što su označavanje sintakse, podudaranje zagrada i Intellisense, i ima podršku za mnoge različite jezike. Ovi jezici uključuju TypeScript, JavaScript, JSON, HTML, CSS, C#, C++ i još mnoge druge, što ga čini idealnim za TypeScript razvoj u veb stranicama ili Nodeu. Glavni fokus Visual Studio Codea je trenutno na ASP.NET razvoju pomoću jezika C# i Node razvoju pomoću TypeScripta.

Instaliranje VSCodea

VSCode može da se instalira na Windows sistem jednostavnim preuzimanjem i pokretanjem instalera. Na Linux sistemima VSCode je obezbeđen kao `.deb` paket, kao `.rpm` paket ili kao binarni tar fajl. Za Mac sistem preuzmite `.zip` file, raspakujte ga, pa kopirajte fajl Visual Studio Code.`app` u direktorijum aplikacija.

Istraživanje VSCodea

Kreirajte novi direktorijum za skladištenje izvornog koda i pokrenite VSCode. To možete da uradite tako što ćete potražiti direktorijum i izvršiti komandu `code.` iz komandne linije. Na Windows sistemima pokrenite VSCode, pa selektujte **Select File | Open** folder iz linije menija. Pritisnite prečicu *Ctrl–N* da biste kreirali novi fajl i ukucajte sledeće:

```
console.log("hello  vscode");
```

Vidite da u ovoj fazi nema isticanja sintakse, jer VSCode ne može da „zna“ sa kojom vrstom fajla radi. Pritisnite prečicu Ctrl–S da biste snimili fajl i dodelite mu naziv hello.`ts`. Sada, kada VSCode „razume“ da je ovo TypeScript fajl, biće vam dostupne Intellisense funkcije i označavanje sintakse.

Kreiranje fajla tasks.json

Prečica na tastaturi za izgradnju projekta u VSCodeu je Ctrl–Shift–B. Ako pokušamo da izgradimo projekat u ovoj fazi, VSCode će prikazati poruku `No task runner configured` i pružiće nam opciju **Configure Task Runner**. Možemo da izaberemo koju vrstu pokretača zadataka ćemo da konfigurišemo, uključujući Grunt, Gulp i veliki broj drugih opcija. Selektovanje jedne od ovih opcija će automatski kreirati fajl `tasks.json` u direktorijumu .vscode i otvoriće ga za editovanje.

Kao primer, selektujmo opciju **TypeScript – tsconfig.**`json`. Izvršićemo jednu promenu na generisanom `tsconfig.json` fajlu i podesićemo vrednost opcije `"showOutput"` na `"always"`, umesto `"silent"`. To će nametnuti VSCodeu obavezu da otvori izlazni prozor kad god vidi probleme u kompajliranju.

Fajl `tasks.json` sada sadrži sledeće:

```
// A task runner that calls the Typescript compiler (tsc) and
// compiles based on a tsconfig.json file that is present in
// the root of the folder open in VSCode
{
  "version": "0.1.0",
  "command": "tsc",
  "isShellCommand": true,
  "showOutput": "always",
  "args": ["-p", "."],
  "problemMatcher": "$tsc"
}
```

Izgradnja projekta

Primer projekta sada možemo da izgradimo pritiskom na prečicu *Ctrl–Shift–B*. Vidite da u osnovnom direktorijumu projekta imamo, kao rezultat koraka kompajliranja, fajlove `hello.js` i `hello.js`.map.

Kreiranje fajla launch.json

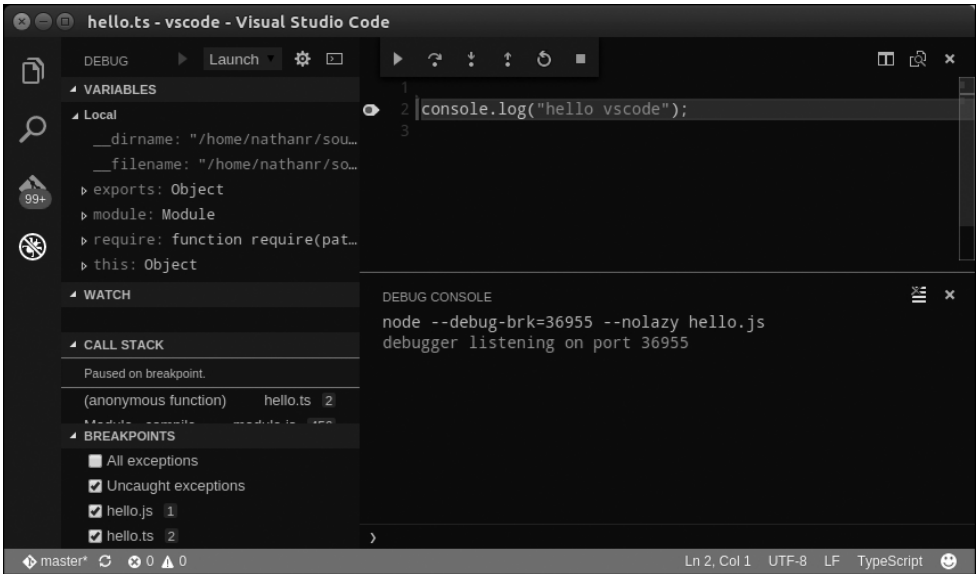
VSCode uključuje integrisani program za ispravljanje grešaka, koji može da se upotrebi za ispravljanje grešaka u TypeScript projektima. Ako otvorimo panel **Debugger** ili jednostavno pritisnemo prečicu *F5* da bismo započeli ispravljanje grešaka, VSCode će zatražiti da selektujemo okruženje za ispravljanje grešaka. Selektovaćemo opciju **Node.js** koja će kreirati fajl launch.`json` u direktorijumu .vscode i otvoriće ga za editovanje. Potražićemo opciju pod nazivom `"program"` i modifikovaćemo je na `"${workspaceRoot}/hello.js"`. Pritisnite ponovo prečicu *F5* i VSCode će pokrenuti fajl `hello.js` kao Node program i ispisaće rezultate u prozor za ispravljanje grešaka:

```
node  --debug-brk=34146 --nolazy hello.js
debugger listening on  port  34146
hello  vscode
```

Podešavanje tačaka prekida

Upotreba tačaka prekida i ispravljanje grešaka u ovoj fazi će funkcionisati samo na generisanim .js JavaScript fajlovima. Izvršićemo jednu promenu na generisanom `tsconfig.json` fajlu i podesićemo vrednost opcije `"showOutput"` na `"always"`, umesto `"silent"`. To će nametnuti VSCodeu obavezu da otvori izlazni prozor kad god vidi probleme u kompajliranju.

Sada možemo da podesimo tačke prekida direktno u `.ts` fajlovima koje će koristiti VSCode funkcija za ispravljanje grešaka:



Ispravljanje grešaka Node aplikacije unutar Visual Studio Codea

Ispravljanje grešaka na veb stranicama

Ispravljanje grešaka TypeScripta koji se pokreće unutar veb stranice u VSCodeu zahteva još malo podešavanja. VSCode koristi Chrome funkciju za ispravljanje grešaka za dodavanje pokrenutoj veb stranici. Da bismo omogućili ispravljanje grešaka na veb stranicama, prvo treba da modifikujemo fajl launch.`json` i da dodamo nove opcije pokretanja na sledeći način:

```
"configurations": [
  {
    "name": "Launch",
    ...
  },
  {
```

```
      "name": "Attach 9222",
      "type": "chrome",
      "request": "attach",
      "port": 9222,
      "sourceMaps": true
    }
  ]
```

Ova opcija pokretanja je nazvana `"Attach 9222"` i biće dodata u pokrenutu instancu Chromea pomoću porta za ispravljanje grešaka `9222`. Snimite launch.`json` fajl i kreirajte HTML stranicu pod nazivom `index.html` u osnovnom direktorijumu projekta na sledeći način:

```
<html>
  <head>
  <script src="helloweb.js"></script>
  </head>
  <body>
    hello vscode
    <div id="content"></div>
  </body>
</html>
```

Ovo je veoma jednostavna stranica, koja učitava fajl `helloweb.js` i prikazuje tekst `hello vscode`. Fajl `helloweb.ts` je sledeći:

```
window.onload = () => {
  console.log("hello vscode");
};
```

Ovaj TypeScript kod jednostavno čeka da se učita veb stranica, nakon čega u konzoli ispisuje „`hello vscode`“.

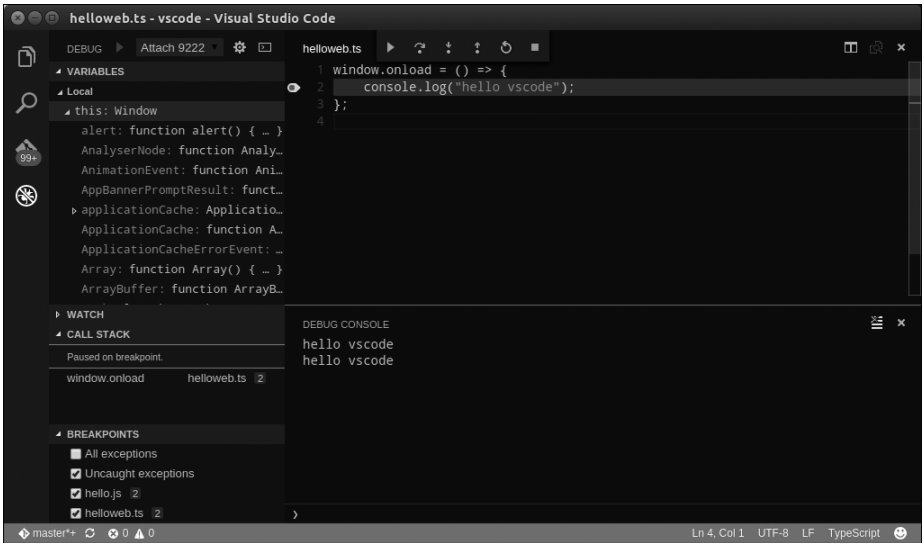
Sledeći korak je pokretanje Chromea pomoću opcije porta za ispravljanje grešaka. Na Linux sistemima to možete da izvršite iz komandne linije na sledeći način:

```
google-chrome --remote-debugging-port=9222
```

Treba da obezbedimo da ne postoje druge instance Chromea koje su pokrenute da bismo ga upotrebili za ispravljanje grešaka pomoću VSCodea.

Zatim ćemo učitati fajl `index.html` u pretraživaču, koristeći sintaksu `file://<full_path_to_file>/index.html`. Trebalo bi da možete da vidite HTML fajl kako renderuje tekst „`hello vscode`“.

Sada ćemo se vratiti u VSCode - kliknućemo na ikonicu za ispravljanje grešaka i selektovaćemo opciju **Attach 9222** u padajućem meniju pokretača. Kada pritisnete prečicu F5, trebalo bi da VSCode funkcija za ispravljanje grešaka bude dodata u pokrenutu instancu Chromea. Zatim je potrebno „osvežiti“ stranicu u Chromeu da bi bilo započeto ispravljanje grešaka:



Ispravljanje grešaka na veb stranici u Visual Studio Codeu

Malim izmenama u fajlu launch.`json` možemo da kombinujemo ove ručne korake u jedan pokretač na sledeći način:

```
{
    "name": "Launch chrome", "type": "chrome", "request" :
    "launch",
    "url" : "file:/// … insert full path here … /index.html",
    "runtimeArgs": [
        "--new-window",
        "--remote-debugging-port=9222"
    ],
    "sourceMaps": true
}
```

U ovoj konfiguraciji pokretanja promenili smo svojstvo `request` sa `"attach"` na `"launch"`, što će pokrenuti novu instancu Chromea i automatski otvoriti putanju fajla specifikovanu u svojstvu `"url"`. Svojstvo `"runtimeArgs"` sada specifikuje udaljeni port za ispravljanje grešaka `9222`. Kada je ovaj pokretač postavljen, možemo da pritisnemo F5 da bismo pokrenuli Chrome, koristeći odgovarajući URL i opcije za ispravljanje grešaka HTML aplikacija.

Ostali editori

Postoji veliki broj editora koji uključuju podršku za TypeScript, kao što su Atom, Brackets, pa, čak, i stari Vim. Svaki od njih ima različite nivoe podrške TypeScripta, uključujući označavanje sintakse i Intellisense. Upotreba ovih editora predstavlja samu osnovu TypeScript razvojnog okruženja, gde se oslanjamo na komandnu liniju za automatizovanje zadataka izgradnje. Editori nemaju ugrađene alatke za ispravljanje grešaka i stoga se ne kvalifikuju kao **integrisano razvojno okruženje (IDE**), ali mogu da se upotrebe za izgradnju TypeScript aplikacija. Osnovni tok rada korišćenjem ovih editora je sledeći:

- kreiranje i modifikovanje fajlova pomoću editora
- pozivanje TypeScript kompajlera iz komandne linije
- pokretanje ili ispravljanje grešaka u aplikaciji pomoću postojećih funkcija za ispravljanje grešaka

Upotreba Grunta

U osnovnom okruženju sve promene u TypeScript fajlu izvršićemo ponovnom upotrebom komande `tsc` iz komandne linije svaki put kada želimo da kompajliramo projekat. Očigledno je veoma dosadno prebacivati se u komandnu liniju i ručno kompajlirati projekat kad god se izvrši neka promena. U ovim situacijama pomažu nam alatke kao što je Grunt - on je automatizovani pokretač zadataka (http://gruntjs.com), koji može da automatizuje mnoge dosadne zadatke kompajliranja, izgradnje i testiranja. U ovom odeljku ćemo upotrebiti Grunt za posmatranje TypeScript fajlova i automatsko pozivanje `tsc` kompajlera kada je fajl snimljen.

Grunt se pokreće u Node okruženju i, stoga, treba da bude instaliran kao `npm` zavisnost projekta. Ne može da bude instaliran globalno kao većina `npm` paketa. Da bismo ga instalirali, potrebno je da kreiramo fajl `packages.json` u osnovnom direktorijumu projekta. Otvorićemo komandnu liniju, pronaći osnovni direktorijum projekta i, jednostavno, ukucati:

```
npm  init
```

Zatim, pratimo zahteve za unos. Možemo da ostavimo većinu opcija na standardnim podešavanjima i uvek možemo da se vratimo da bismo, u slučaju da treba da izvršimo neke promene, editovali fajl `packages.json` koji je kreiran u ovom koraku.

Sada možemo da instaliramo Grunt. On ima dve komponente, koje treba da budu instalirane nezavisno. Prvo, treba da instaliramo Grunt interfejs komandne linije koja omogućava da pokrenemo Grunt iz komandne linije. To možemo izvršiti na sledeći način:

```
npm  install –g  grunt-cli
```

Druga komponenta je instaliranje Grunt fajlova unutar direktorijuma projekta:

```
npm  install grunt  --save-dev
```

Opcija --save-dev će instalirati lokalnu verziju Grunta u direktorijum projekta, pa će i drugi projekti na sistemu moći da koriste različite verzije Grunta. Takođe su nam potrebni paketi `grunt-exec` i `grunt-contrib-watch`, koji mogu da budu instalirani pomoću sledećih komandi:

```
npm  install grunt-exec --save-dev
npm  install grunt-contrib-watch --save-dev.
```

Na kraju, biće nam potreban `GruntFile.js`. Pomoću editora kreiraćemo novi fajl, snimićemo ga kao `GruntFile.js` i unećemo sledeći JavaScript. Imajte na umu da ovde kreiramo JavaScript fajl, a ne TypeScript fajl. Možete da pronađete kopiju ovog fajla u primeru izvornog koda koji je povezan sa ovim poglavljem:

```
module.exports = function (grunt) {
  grunt.loadNpmTasks('grunt-contrib-watch');
  grunt.loadNpmTasks('grunt-exec');
  grunt.initConfig( {
    pkg: grunt.file.readJSON('package.json'),
    watch : {
        files : ['**/*.ts'],
        tasks : ['exec:run_tsc']
    },
    exec: {
        run_tsc: { cmd : 'tsc'}
    }
  });
  grunt.registerTask('default', ['watch']);
};
```

Ovaj `GruntFile.js` sadrži jednostavnu funkciju za pokretanje Grunt okruženja i specifikuje komande za pokretanje. Prve dve linije funkcije su učitavanje paketa `grunt-contrib-watch` i `grunt-exec` kao `npm` zadataka. Pozvaćemo funkciju `initConfig` za konfigurisanje zadataka za pokretanje. Ovaj odeljak konfiguracije ima svojstva `pkg`, `watch` i `exec`. Svojstvo `pkg` se koristi za učitavanje fajla `package.json`, koji smo ranije kreirali kao deo koraka `npm` init.

Svojstvo `watch` ima dva podsvojstva. Svojstvo files specifikuje šta se traži (u ovom slučaju to su `.ts` fajlovi u izvornom stablu), a niz `tasks` specifikuje da bi trebalo da pokrenemo komandu `exec:run_tsc` kada je fajl promenjen. Na kraju, pozvaćemo funkciju `grunt.registerTask` specifikovanjem standardnog zadatka posmatranja promene fajla.

Sada možemo da pokrenemo Grunt iz komandne linije na sledeći način:

```
grunt
```

Kao što možete da vidite iz komandne linije, Grunt je pokrenuo zadatak `watch` i čeka promene u bilo kom `.ts` fajlu:

```
Running "watch" task
Waiting...
```

Otvorite bilo koji TypeScript fajl i izvršite malu promenu (dodajte razmak ili nešto slično), pa pritisnite prečicu Ctrl-S da biste snimili fajl. Sada proverite ispis u Grunt komandnoj liniji. Trebalo bi da vidite nešto slično sledećem:

```
>> File  "hellogrunt.ts" changed.
Running "exec:run_tsc" (exec) task
Done,  without errors.
Completed in 2.008s at Sat Mar 19 2016 20:27:17 GMT+0800 (W.
Australia Standard Time)  - Waiting...
```

Ovaj ispis komandne linije je potvrda da je posmatranjem Grunta identifikovano da je fajl `hellogrunt.ts` promenjen, da je pokrenut zadatak `exec:run_tsc` i da Grunt čeka sledeću promenu u fajlu. Takođe bi trebalo da vidite fajl `hellogrunt.js` u istom direktorijumu u kojem se nalazi i Typescript fajl.

REZIME

U ovom poglavlju saznali ste šta je TypeScript i koje prednosti može da uvede u JavaScript razvoj. Takođe smo opisali podešavanja razvojnog okruženja, koristeći neke popularne IDE-ove, i videli ste kako izgleda osnovno razvojno okruženje. Sada, kada je razvojno okruženje podešeno, možemo da započnemo malo detaljniji pregled samog TypeScript jezika. U sledećem poglavlju ćemom započeti pregled tipova, pa promenljivih, a zatim ćemo opisati funkcije.