Javascript transpileri

Grubišić, Zvonimir

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:517250

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-01-02**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski studij

JAVASCRIPT TRANSPILERI

Završni rad

Zvonimir Grubišić

Osijek, 2017

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 23.09.2018.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Zvonimir Grubišić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3637, 29.09.2017.
OIB studenta:	50946540896
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Hrvoje Leventić
Sumentor iz tvrtke:	
Naslov završnog rada:	Javascript transpileri
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	23.09.2018.
Datum potvrde ocjene Odbora:	26.09.2018.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 01.10.2018.

Ime i prezime studenta:	Zvonimir Grubišić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3637, 29.09.2017.
Ephorus podudaranje [%]:	1%

Ovom izjavom izjavljujem da je rad pod nazivom**: Javascript transpileri**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD ... 1
1.1. Zadatak završnog rada ... 2
2. JAVASCRIPT ... 3
2.1. Općenito ... 3
2.2. Javascript pokretači ... 3
2.3. ECMAScript ... 4
3. TRANSPILERI ... 6
3.1. Babel ... 6
3.2. CoffeeScript ... 10
4. TYPESCRIPT ... 13
4.1. Uvod u TypeScript ... 13
4.2. TypeScript - objektno orijentirani jezik ... 13
4.3. Usporedba TypeScripta i Javascripta ... 16
5. APLIKACIJA ZA ODABIR TERMINA DEMONSTRATURE ... 20
5.1. Opis aplikacije ... 20
5.2. Opis uloga ... 20
5.3. Korisničko sučelje ... 21
5.4. Baza podataka ... 28
6. ZAKLJUČAK ... 30
LITERATURA ... 31
SAŽETAK ... 33
ABSTRACT ... 34
ŽIVOTOPIS ... 35
PRILOZI ... 36

1. UVOD

Transpileri (engl. *transpilers*) se također nazivaju *source-to-source* prevoditelji, tj. prevoditelji izvornog u izvorni kod [2]. Ono što transpileri zapravo čine je pretvaranje, tj. prevođenje jednog programskog jezika u drugi programski jezik. Ovaj rad je usredotočen na Javascript transpilere za koje se kaže da ciljaju Javascript programski jezik.

Prvi problem na koji svaki Javascript transpiler nailazi je činjenica da Javascript okruženje može raditi samo sa Javascriptom. Naime Javascript kao programski jezik je vrlo raširen te postoje mnoge različite verzije i njegove implementacije. Standard na kojem se zasnivaju sve verzije Javascripta naziva se ECMAScript i njegova najnovija verzija je ES 2017. Najpoznatija implementacija ECMAScripta je Javascript te se često ta verzija i naziva „obični“ ili „čisti“ (engl. *vanila*) Javascript.

Dakle ako se u konzolu u pregledniku pokuša unijeti program napisan u „neprevedenom“, tj. izvornom transpileru, ili čak napisan u ES2017[1], rezultat će biti pogreške. [2] Ali ako se taj isti kod prvo provuče kroz prevoditelj transpilera, on će TypeScript, CoffeeScript te ES2017 program pretvoriti u originalni Javascript kod, koji će tek potom, ako je ispravan, biti interpretiran.

Prvo se valja usredotočiti na potrebu transpilera za pisanje običnog Javascripta. Naime, CoffeeScript i TypeScript su lijepi nadodaci, ali ako se koristi obični Javascript, logično je da nema potrebe za nikakvim prevoditeljem, jer bi on, po ranijoj definiciji, prevodio jedan program u taj isti program . Nažalost, nije tako. Za pristup Internetu se koriste različiti preglednici i tako pojedini preglednici koriste svoje Javascript pokretače. Google Chrome koristi *V8*, Mozilla FireFox *SpiderMonkey*, a Internet Explorer *Chakru*. Svaki od tih okružja implementira različite podverzije ES2017[2] rješenja, ima različita svojstva u izvođenju i različitim putevima se približava standardu. Zbog toga će većina običnog Javascript programa raditi u svim najnovijim verzijama Google Chromea, Opere ili Safarija, ali se na starijim verzijama može dogoditi nepredviđeno ponašanje.

Razlike između pojedinih pokretača Javascripta, te samih implementacija Javascripta kao i o ECMAScriptu i što je on uopće objašnjeno je u nastavku rada. Jedan od razloga zbog kojeg koristimo transpilere je činjenica da prevode “običan” Javascript program u onaj koji će raditi na svim preglednicima i njihovim inačicama. [3]

[1] ES2017 - posljednja službena verzija Javascript programskog jezika po ECMA standardima

[2] ECMA International - internacionalna organizacija za izdavanje standarda za informatičke i komunikacijske sustave

1.1. Zadatak završnog rada

Cilj ovog završnog rada je opisati pojam transpilera - što su, čime se bave te kako funkcioniraju. Koja je njihova uloga u modernom web programiranju i zašto su važni. Završni rad će se usredotočiti na nekoliko transpilera - opisani su Babel i CoffeeScript. Nakon njih, najviše pozornosti će biti usmjereno na TypeScript transpiler, koji je korišten u izradi praktičnog dijela završnog rada.

TypeScript će biti detaljno opisan, njegova svojstva kao objektno orijentiranog jezika i novosti koje uvodi u običan Javascript. Također će biti obrađene prednosti u odnosu na običan Javascript. Nakon toga slijedi opis praktičnog dijela aplikacije, podijeljen u četiri manja poglavlja. Prvo će ići generalni opis aplikacije, njena kratka povijest i razlog potrebe za takvom aplikacijom. U aplikaciji postoje 2 različite uloge, te će biti opisane njihove razlike. Najbitniji dio ove aplikacije su korisnici što povlači temu korisničkog iskustva. Ne smije biti izostavljena ni baza podataka pisana u MSSQL-u razrađena za potrebe ove aplikacije.

2. JAVASCRIPT

2.1. Općenito

Javascript je programski jezik više razine, dinamičan, "netipan" i interpretiran pri izvršavanju [1]. Standardiziran je po ECMAScript (više u poglavlju 2.3.) jezičnim usmjerenjima, te zajedno s HTML i CSS programskim jezicima tvori temelj stvaranja internetskih aplikacija i stranica.

Javascript je podržan od strane svih modernih Internet preglednika, te zajedno sa svojim nadodacima (engl. *plug-in*) je jedan od najviše korištenih jezika u izradi internetskog sadržaja.

Kada za neki programski jezik kažemo da je više razine, to znači da su ključne riječi tog jezika slične jeziku ljudi. Ključna riječ je riječ koju programski jezik rezervira jer riječ ima posebno značenje. Ključne riječi mogu biti naredbe ili parametri. Svaki programski jezik ima skup ključnih riječi koje se ne mogu koristiti kao nazivi varijabli. Ključne riječi se ponekad nazivaju rezervirana imena.

Druga bitna značajka Javascripta je da je dinamičan [13]. Odlika dinamičnih programskih jezika je da se prije izvršavanja ne prevode i da prilikom izvršavanja programa oblik jezika je vrlo sličan ili isti onom kakav je prije pokretanja programa - to omogućava programeru, tj. osobi koja piše program, da lakše uvrsti nove promjene i bolje shvati u kojem djelu programskog koda se program trenutno nalazi. Ovo također znači da sve varijable moraju biti navedene, tj. inicijalizirane prije korištenja.

Kada za neki programski jezik kažemo da je netipan, to znači da prilikom inicijalizacije neke varijable, tj. prilikom zauzimanja memorije za pojedini podatak ne moramo računalu napomenuti kojeg je tipa taj podatak [13]. Što to konkretno znači je da ako u neku varijablu, tj. na neko mjesto u memoriji želimo spremiti podatak, dovoljno je da koristimo ključnu riječ `var`. Tokom pisanja programa, ta varijabla može sadržavati podatke različitih tipova (cjelobrojni, *string*, klasu), a kojeg je točnog tipa prepoznaje interpreter prilikom izvršavanja programa.

2.2. Javascript pokretači

Javascript pokretači (engl. *engines*) je naziv za programe ili interpretere koji pokreću Javascript programski jezik. Pokretač može biti tradicionalni interpreter ili koristiti Just in Time (JIT - engl. *Just In Time* - koncept izvršavanja programa u realnom vremenu) prevođenje. Glavna svrha Javascript pokretača je prebaciti program koji programer napiše u Javascriptu u još brži i optimiziraniji za pojedini preglednik ili aplikaciju u kojoj se koristi. Svaki pokretač implementira određenu verziju ECMAScript-e i dizajniran je za rad s određenim preglednikom ili okruženjem za izvršavanje (i pokretanje). Neki od najpoznatijih pokretača su V8, JavascriptCore, SpiderMonkey i Chakra. [17]

V8 je Javascript pokretač koji se koristi u Google Chrome pregledniku. Napisan je u programskom jeziku C++. Osim što pokreće Javascript programski jezik, također brine o alokaciji (popunjavanju) memorije te o "skupljanju smeća" (engl. *garbage collector*). U samom pokretaču nalaze se dva prevoditelja:

- Full-codegen - brzo prevodi, ali stvara program koji je loše optimiziran
- Crankshaft - prevodi sporije, ali zato stvara visoko optimizirani program

Način na koji ova dva prevoditelja surađuju je također poseban. U slučaju da Crankshaft prepozna da Full-codegen nije dovoljno optimizirao program, zamjenjuje ga bolje optimiziranim programom. Taj proces naziva se *crankshafting*[3]. [17]

Javascript Core je pokretač korišten u pregledniku WebKit. Originalni Javascript program prolazi kroz ove korake:

1. Pokretač izvodi leksičku analizu programa, gdje razdvaja naredbe programa u seriju *stringova* (engl. izraz za redak teksta u kontekstu programa) ili tokena s računalu jasnim značenjem
2. Tokeni su ponovno analizirani i smješteni su u sintaktičko stablo
3. Četiri JIT procesa ponovno analiziraju te izvode program koji je sada preveden u *bytecode* - program napisan u bajtovima. [17]

Spidermonkey je pokretač koji koristi preglednik Mozilla Firefox. Dolazi sa ugrađenom ljuskom (engl. *shell*) koji korisniku nudi pregršt naredbi i opcija za upravljanje, prevođenje te naravno pokretanje samog Javascript programa. Sam pokretač sadržava prevoditelj, pokretač programa i *garbage collector*. Prolazi kroz Javascript bytecode instrukciju po instrukciju. [18]

Chakra je Javascript pokretač koji je korišten u Microsoftovom pregledniku Edge te u ostalim Windows aplikacijama i programima koje su napisane u HTML-u, CSS-u i naravno, Javascriptu. Podržava JIT prevođenje za x86/x64/ARM strukture procesora, *garbage collection* i druge nove mogućnosti programskog jezika Javascript. [19]

2.3. ECMAScript

Prije prve službene verzije, ime programskog jezika Javascript bio je LiveScript, ali je iz marketinških razloga preimenovan u Javascript, kako bi ime asociralo na tada vrlo popularni jezik Java.

Prva verzija Javascripta službeno je izašla u ožujku 1996. godine. Podržavali su je preglednici Netscape Navigator 2.0 i Internet Explorer 3.0. U studenom 1996. uz pomoć ECMA International stvoren je standard imenom ECMAScript, a Javascript njegova najpoznatija implementacija. [16]

Od ostalih implementacija bitno je spomenuti JScript i ActionScript. JScript je implementacija ECMAScripta koju je stvorio Microsoft za pisanje skripti za server na njihovom IIS (*Internet Information Server*) okruženju. ActionScript je razvio Macromedia Inc., tj. kasnije

[3] Engleski izraz direktno od izvora

Adobe Systems za aplikacije koje koriste njihovu Adobe Flash Player platformu. Razvoj Javascripta danas uglavnom vode Netscape i Mozilla.

Do prosinca 1999., službeno su izašle 3 verzije ECMAScripta. One podržavaju regularne izraze (engl. *regex*), bolje upravljanje *stringovima*, rad s iznimkama, ispis različitih brojevnih formata i drugo. Službena verzija 4 je preskočena zbog velikih razlika u razvoju te tehničkih poteškoća. Neka svojstva su integrirana u verziju 5.U prosincu 2009. stvorena je verzija 5 ECMAScripte. Ona dodaje strogi način, razjašnjava različite nejasnoće i ubacuje postavljače i dobavljače (engl. *getters* i *setters*), podršku za JSON oblik podatka te refleksiju, tj. čitanje vrijednosti svojstava objekata preko imena svojstava. U lipnju 2011. izlazi ECMAScript 5.1 koja podržava ISO/IEC 16262 internacionalni standard. U 2015. Izlazi ECMAScript 2015. Language. Ta verzija je poznata pod nazivima ES2015 i ES6 [21]. Ubacuje podršku za klase i module te novu sintaksu koja pomaže pri stvaranju kompleksnijih aplikacija.

Sljedeće 2 godine izlaze nove verzije ECMAScript-a koje slijede isti oblik imenovanja. Trenutno najnovija važeća verzija je ECMAScript 2017.O detaljnim razlikama između pojedinih okružja Javascripta, tj. preglednika i koje standarde koriste, može se vidjeti u tablici o usuglašenosti ECMAScripte - Prilog 2.1.

3. TRANSPILERI

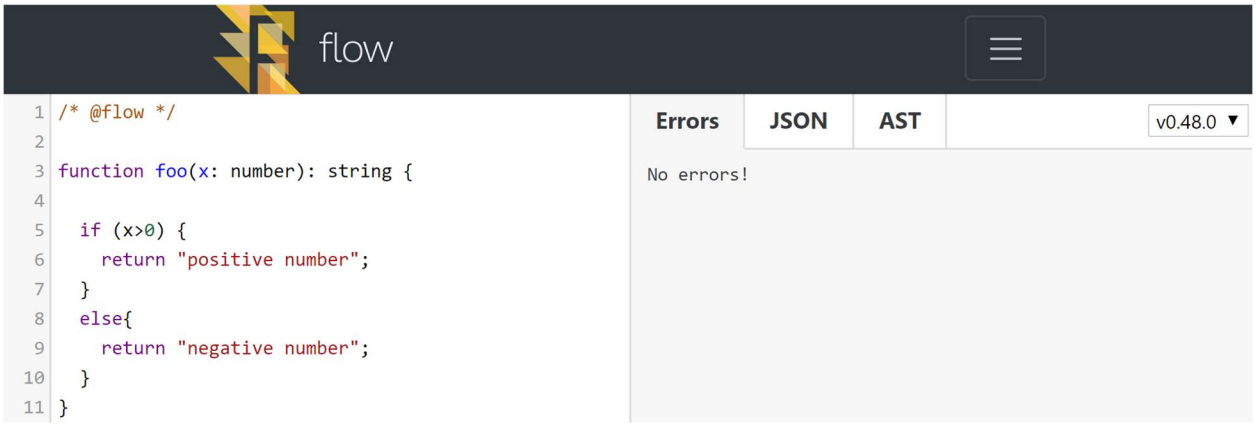
Najpoznatiji transpileri su TypeScript, Babel, Traceur, Clojure, Dart, Elm, Haxe, ScalaJs. Postoji još drugih, manje poznatih transpilera.

Ovdje se pojavljuju pojmovi TypeScript i CoffeeScript. CoffeeScript uvodi bolju sintaksu za brojna rješenja koja nisu dostupna u "običnom" Javascriptu i pritom uklanja neke druge loše dijelove. TypeScript djeluje više drastično te u Javascript uvodi objektno orijentirano programiranje.

Jedan način gledanja transpilera je taj da nadograđuju Javascript, dodavajući u njega inače nepodržane ili teže dostupne mogućnosti. Korištenjem transpilera, programeri mogu koristiti nove značajke i aplikacijske programe sučelja (engl. *Application programming interface* - API) točno kako su pisani, što dalje unapređuje kvalitetu programa. Transpileri također pozitivno utječu na učinkovitost pisanja novih programa, ali i pronalaska grešaka čak i u starim programima (u slučaju prepisivanja programa iz izvornog Javascripta u neki transpiler).

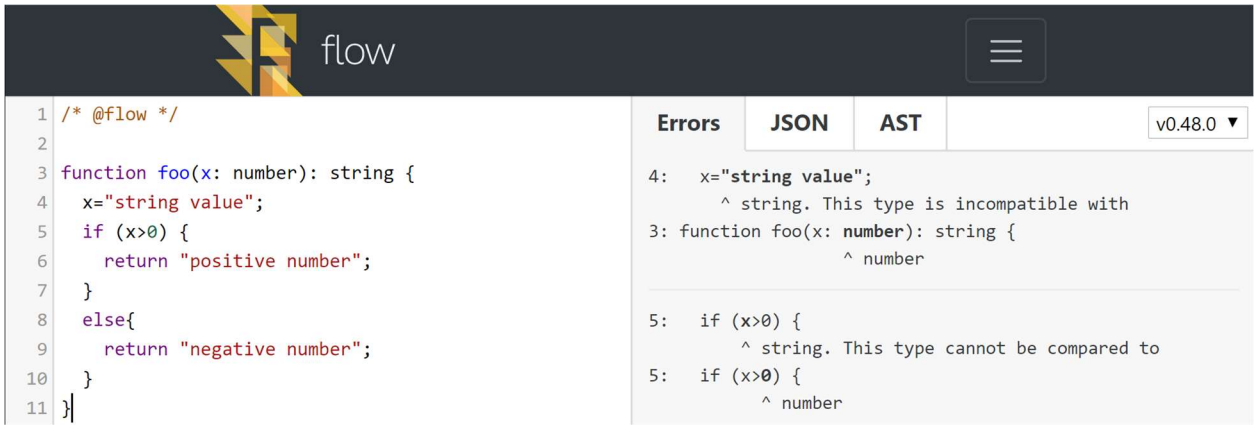
3.1. Babel

Babel je transpiler za Javascript koji može prevesti ES6 verziju i sve nove verzije Javascripta u program koji može pokrenuti većina današnjih internetskih preglednika. Kreirao ga je Australski programer Sebastian McKenzie s idejom podržavanja nove sintakse koja dolazi u ES6 te s ugrađenom podrškom za *React JSX* nadodatke te *Flow anotacije*[4]. Na Slici 3.1. i Slici 3.2. možemo vidjeti primjer programa s anotacijom u slučaju točnog programa i programa s greškom.[5]



Sl. 3.1.: Flow anotacija

[4] Pomoćni savjeti koji se prikazuju programeru prilikom pisanja koda, najčešće u obliku iskočnih prozora i padajućih izbornika gdje se predlaže korištenje postojećih datoteka, varijabli ili knjižnica



Sl. 3.2.: Flow anotacija

Babel se često uspoređuje s CoffeeScriptom, najčešće zbog njihove slične sintakse i činjenice da su oba značajno oblikovali razvoj ES6 standarda. Bitna razlika između CoffeeScripta i Babela je činjenica da Babel nije "samo" programski jezik, već čitava platforma. Naime, Babel prati najnoviju verziju Javascripta, tj. podržava sve novosti upoznate u novim verzijama ECMAScript standarda. Između ostalog, podržava i asinkrone[5] operacije kroz ključne riječi *async/await*. Pomoću njih programeri mogu puno jednostavnije razvijati program koji ne ovisi o *callback* (engl. pozvati nazad) metodama. Ovdje je Babel napredniji nekoliko godina od mnogih internet preglednika. [4]

Babelovo tajno oružje su nadodaci: Babel omogućuje da prilagođeni transformatori programa budu uključeni u proces prevođenja. Ovi transformatori uzimaju AST (engl. *Abstract Syntax Tree* - AST) i obavljaju manipulacije na njemu prije nego što se pretvori u izvršni Javascript. AST je zbirka ugniježđenih objekata koji opisuju strukturu programa. Manipulacije nad AST omogućuju dodavanje provjera prije i tokom vremena izvršavanja korištenjem *Flow* anotacija - Slike 3.1. i 3.2., kao i uklanjanje nepotrebnih zatvaranja (optimizaciju programa)- Slika 3.3. [4]

[5] Operacije koje međusobno ne čekaju kraj izvođenja, već se izvode paralelno

Pretvara ovakav kod:

```
function demo (input) {
  return input.map(item => item + 1).map(item => item + 2);
}
```

U ovakav kod:

```
function _ref(item) {
  return item + 1;
}

function _ref2(item) {
  return item + 2;
}

function demo(input) {
  return input.map(_ref).map(_ref2);
}
```

Sl. 3.3.: Babelovo poboljšanje programa

Ostale bitne mogućnosti Babela su praćenje jednog opsega programa (engl. *scope*), tj. memorije programa kao i preimenovanje varijabli te još mnoštvo drugih mogućnosti koje pojednostavljuju kompliciranе AST transformacije. [4]

Jedan kompromis koja sva Javascript okruženja moraju napraviti je vrijeme prevođenja i vrijeme optimizacije. Na webu je iznimno bitno da se Javascript datoteke učitavaju i izvršavaju brzo, inače je korisničko iskustvo loše. JIT prevođenje jednostavno nema vremena za obavljanje svih optimizacija koje su tehnički moguće tijekom izvođenja, pa stoga mora napraviti preinake, izbjegavajući određene operacije koje bi omogućile brži program, ali su previše "skupe" za izvođenje. Također ne može izvršiti mnogo optimizacija zbog dinamičnog sustava Javascripta. Umjesto toga mora obavljati neke optimizacije, ali u isto vrijeme imati mogućnost vratiti se na sporiji i sigurniji program ako se neke pretpostavke pokazuju pogrešnima. Ovakav način programiranja i pokretanja Javascript programa iznimno je skup i može vrlo loše utjecati na izvedbu.

Budući da se izvodi tokom prevođenja, Babel nema takvih intenzivnih vremenskih ograničenja. Zbog impresivnog praćenja opsega i tipa varijabli, on je sjajan temelj na kojem se mogu graditi transformatori koji obavljaju takve optimizacije.

Primjeri optimizacije su zamjena izraza tipa `1+2+3` rezultatom `6`. Prikazano na Slici 3.4..

```
function add (a, b) {
  return a + b;
}
function demo () {
  return add(1, 2);
}
console.log(demo());
```

treba biti pretvoreno u:

```
function demo () {
  return 1 + 2;
}
console.log(demo());
```

što se dalje pretvara u:

```
function demo () {
  return 3;
}
console.log(demo());
```

i konačno u:

```
console.log(3);
```

Sl. 3.4.: Babelova optimizacija programa

Još jedna od optimizacija je izbacivanje konstantnih, tj. nevarijabilnih izraza iz petlji. Izraz `let tau = pi * 2` se ne mijenja u nijednoj iteraciji petlje te može biti izbačen. Rezultati su vidljivi na Slici 3.5..

```
for (let i = 0; i < 10; i++) {
  let tau = pi * 2;
  foo(tau * i);
}
```

se pretvara u :

```
let tau = pi * 2;
for (let i = 0; i < 10; i++) {
  foo(tau * i);
}
```

Slika 3.5.: Babelova optimizacija programa - izbacivanje nevarijabilnih izraza

Babel, za razliku od pokretača V8 korištenog u pregledniku Google Chrome, optimizira preko “otvaranja petlji” (engl. *loop unrolling*). Što je to točno je vidljivo na Slici 3.6..

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

se pretvara u:

```
console.log(0);
console.log(1);
console.log(2);
console.log(3);
console.log(4);
console.log(5);
console.log(6);
console.log(7);
console.log(8);
console.log(9);
```

Sl. 3.6.: Babelova optimizacija programa

Babel će vjerojatno u skoroj budućnosti postati glavni pokretač ES6 + ESNext usvajanja i pomagati u pisanju modernih i brzih Javascript programa. Njegov sustav dodataka razlikuje ga od konkurenata kao što su CoffeeScript i TypeScript i ima potencijal za pokretanje nove inovacije u Javascriptu. [4]

3.2. CoffeeScript

CoffeeScript koristi snažne značajke Javascript bez učenja nekih manje poznatih značajki istoga. dodavanjem "sintaktičkog šećera" (engl. *syntactic sugar*) u Javascript. CoffeeScript nema točka-zareze i vitičaste zagrade, te ima sličnu sintaksu programima Pythonu i Rubyju što znači da CoffeeScript razvojni programeri pišu kraće programe i tako ih pišu brže. Ovo također olakšava čitanje i održavanje. Zbog optimizacije CoffeeScript program nakon prevođenja postaje jednako uspješan ili u nekim slučajevima još učinkovitiji Javascript program. [6]

```
var square = function(x) {
  return x * x;
}
```

```
square = (x) ->
  x * x
```

Sl. 3.7.: Primjer CoffeeScript (gornji) i ekvivalentnog Javascript programa (donji)

CoffeeScript je vrlo jezgrovit i uzima u obzir razmake, tj. takozvani "bijeli prostor" (engl. *white space*). Ovo može smanjiti program za trećinu ili čak polovinu originalnog Javascript-a. Osim toga, CoffeeScript ima neke uredne značajke, kao što su razvrstavanja polja, aliasi prototipa koji dodatno smanjuju količinu napisanog programa.

Jedan od nedostatka korištenja CoffeeScripta je što uvodi još jedan korak između programera i Javascripta. CoffeeScript taj problem rješava što je najbolje moguće stvaranjem čistog i čitljivog Javascripta i sa svojim integracijama pokretača koji automatiziraju prevođenje. Drugi nedostatak, kao i kod bilo kojeg novog jezika, jest činjenica da je zajednica u ovom trenutku relativno malena, a teško je pronaći suradnike koji već znaju jezik. No, CoffeeScriptu raste popularnost. [7]

CoffeeScript nije ograničen na preglednik i može se koristiti na mnogim Javascript poslužiteljskim (engl. *client*) implementacijama, kao npr. Node.js. Osim toga, CoffeeScript dobiva puno širu upotrebu i integraciju. Integriran je u program RubyOnRails od verzije 3.1. (sredina 2011. godine). U vrijeme pisanja ovog članka, najnovija verzija Rails-a je 5.2.1.. [20]

Bitna razlika između CoffeeScripta i običnog Javascripta je što određene ključne riječi, `function` i `var` nisu dopuštene. Dakle, prilikom pisanja CoffeeScript programa (datoteke s *.coffee* nastavkom), nije moguće koristiti te ključne riječi. Također je uklonjena podrška za globalne varijable. To znači da prilikom inicijalizacije nije potrebno pisati ključnu riječ var, ona će biti sama dodana tokom interpretacije.

Još jedna zanimljiva činjenica vezana za CoffeeScript je da je CoffeeScript tumač (engl. *interpreter*) napisan u CoffeeScript programu.

CoffeeScript uvodi ključnu riječ `not` koja zamjenjuje znak `!` u običnom Javascriptu.

```
if not true then "Panic"
```

Na sličan način se može koristiti i ključna riječ `unless`.

```
unless true "Panic"
```

Korištenje uvjetnog grananja je također drugačije, u slučaju da se koristi grananje u jednom redu. Naime, kako CoffeeScript koristi razmake za prepoznavanje kraja blokova naredbi, potrebno je koristiti ključnu riječ `then`.

```
if a > b then "a is bigger" else "bees"
```

CoffeeScript radi pretvaranje == operatora u `===` i `!=` u `!==`. Razlog ovome je što običan Javascript na taj način uspoređuje tipove i vrijednosti, što često vodi do grešaka u kodu.

CoffeeScript također podržava klase i nasljeđivanje. [8]

Od ECMAScript 5 verzije, uveden je i strogi način (engl. *strict typing*). Strogi način stavlja program u kontekst gdje određene naredbe uzrokuju pojavljivanje upozorenja puno češće nego

inače s ciljem navođenja programera na “pravi put”, tj. sugerirao da odstupaju od najboljih praksi pisanja programa. Ovime se potiče pisanje optimiziranijeg programa, programa s manje grešaka i boljom čitljivosti. CoffeeScript po zadanim pravilima već odgovara većini pravila u strogom načinu. [9]

4. TYPESCRIPT

4.1. Uvod u TypeScript

TypeScript je transpiler za Javascript koji nudi neobavezne statične tipove prilikom pisanja programa, klasa i sučelja. Najveća prednost TypeScripta, osim što u Javascript programski jezik uvodi objektno orijentirano programiranje, što pruža mogućnost bogatijeg razvojnog okruženja gdje je puno lakše uočiti precizne greške prilikom pisanja programa. [10]

TypeScript je javno dostupan program (engl. *open source*) te Apache 2 licenciran i podržava ga Microsoft. Projekt predvodi Anders Hejlsberg, glavni arhitekt programskog jezika C#.

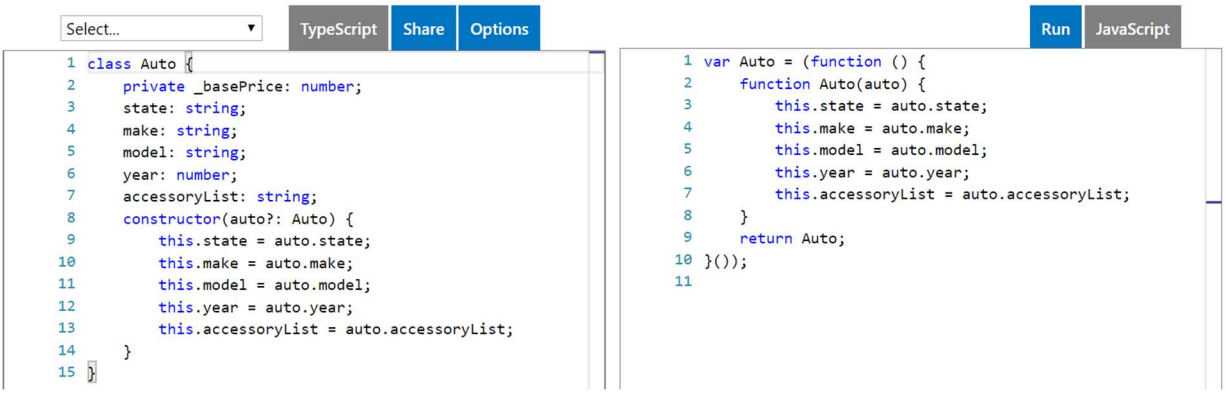
Javascript je, kao što se vidi iz imena, skriptni jezik te kao takav nikad nije dizajniran za stvaranje velikih i složenih aplikacija. Jednostavno ne podržava pisanje dugog i dobro strukturiranog koda. Zato je potreban TypeScript. TypeScript program se prevodi u običan Javascript koji može biti pokrenut na bilo kojem operacijskom sustavu, na bilo kojem poslužitelju te u bilo kojem pregledniku. Bitna stvar koju dopušta TypeScript i njegov prevoditelj je da se u TypeScript programu može pisati i obični Javascript program. [11]

Način na koji TypeScript uvodi koncepte objektno orijentiranog programiranja, tj. klase, sučelja, module i objekte je da se prilikom prevođenja TypeScript program "pretvara" u Javascript program - npr. jedna linija ili ključna riječ se raspisuje u nekoliko novih linija koje postoje u Javascriptu. Upravo zato nema dodatnog troška za memoriju prilikom korištenja TypeScripta. [12]

4.2. TypeScript - objektno orijentirani jezik

TypeScript dakle uvodi koncept objektno orijentiranog programiranja u ne-objektno orijentirani jezik. Prva značajka objektno orijentiranog jezika su naravno klase. Primjer[6] vidimo na Slici 4.1..

[6] Svi primjeri u ovom poglavlju su stranice http://www.typescriptlang.org/play/index.html gdje se može vidjeti pretvaranje TypeScript koda u Javascript kod



Sl. 4.1.: Klasa u TypeScriptu (lijevo) i Javascriptu (desno)

Da bi se neki jezik smatrao objektno orijentiranim, mora podržavati 4 ključna principa [14]:

- Nasljeđivanje
- Apstrakcija
- Enkapsulacija
- Polimorfizam

Nasljeđivanje je koncept koji omogućava jednom objektu, tj. klasi da naslijedi sve osobine i ponašanja njemu roditeljskog objekta, tj. Klase.
Na slikama 4.2. i 4.3. možemo vidjeti nasljeđivanje u TypeScriptu i kako se ono prevodi u Javascript.

Select... TypeScript Share Options

```typescript
class Auto {
    private _basePrice: number;
    state: string;
    make: string;
    model: string;
    year: number;
    accessoryList: string;
    constructor(auto?: Auto) {
        this.state = auto.state;
        this.make = auto.make;
        this.model = auto.model;
        this.year = auto.year;
        this.accessoryList = auto.accessoryList;
    }
}
class Truck extends Auto {
    private _bedLength: string;
    fourByFour: boolean;

    constructor() {
        super();
    }
}
```

Sl. 4.2.: TypeScript kod primjer nasljeđivanja

Run JavaScript

```javascript
var __extends = (this && this.__extends) || (function () {
    var extendStatics = Object.setPrototypeOf ||
        ({ __proto__: [] } instanceof Array && function (d, b) { d.__proto__ = b; }) ||
        function (d, b) { for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p]; };
    return function (d, b) {
        extendStatics(d, b);
        function __() { this.constructor = d; }
        d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());
    };
})();
var Auto = (function () {
    function Auto(auto) {
        this.state = auto.state;
        this.make = auto.make;
        this.model = auto.model;
        this.year = auto.year;
        this.accessoryList = auto.accessoryList;
    }
    return Auto;
}());
var Truck = (function (_super) {
    __extends(Truck, _super);
    function Truck() {
        return _super.call(this) || this;
    }
    return Truck;
}(Auto));

```

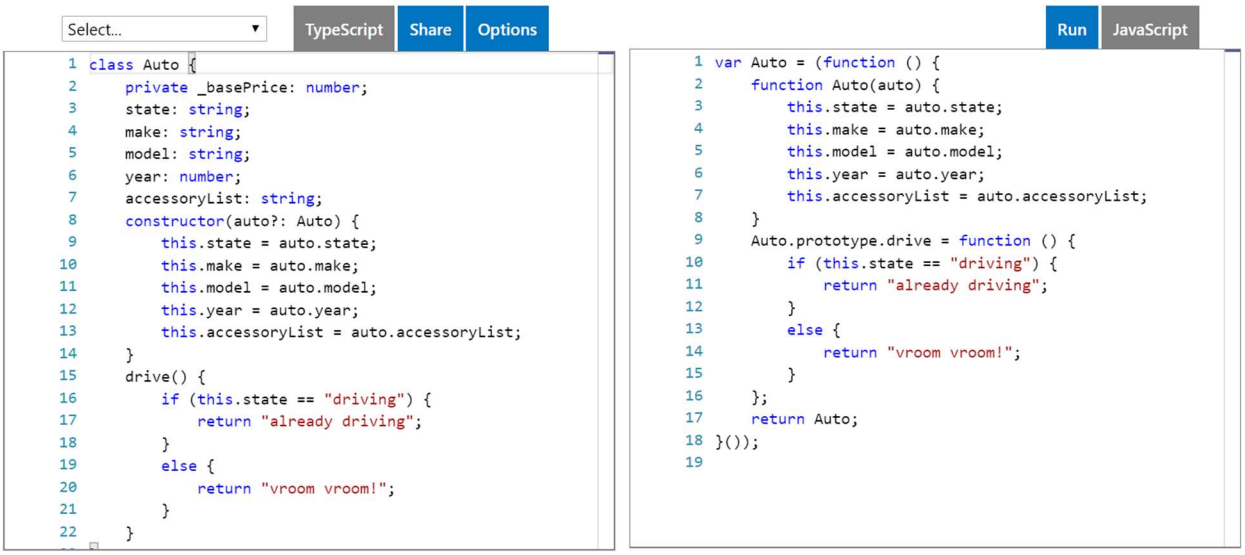
Sl. 4.3.: Javascript kod primjer nasljeđivanja

Apstrakcija je koncept gdje se unutarnji detalji i funkcionalnosti ne prikazuju ostalim dijelovima programa i drugim klasama te objektima. To se najčešće postiže ključnim riječima `public`, `private`, te `protected`. [15]

Apstraktne klase su osnovne klase iz koje se mogu izvesti druge klase. Ne može ih se izravno instancirati. Za razliku od sučelja, apstraktna klasa može sadržavati pojedinosti implementacije za svoje članove. Ključna riječ `abstract` koristi se za definiranje apstraktnih klasa, kao i apstraktnih metoda unutar apstraktne klase.

Polimorfizam je sposobnost nekog objekta da preuzme mnoge oblike. Najčešća upotreba polimorfizma u OOP-u se događa kada se referenca roditeljske klase koristi za označavanje objekta klase djeteta. Svaki objekt koji može proći više "*IS-A*", tj. "*JE LI*" ispita se smatra polimorfnim. U većini objektno orijentiranih jezika, svi jezici su polimorfni jer prolaze "*JE LI*" ispit na tip sebe i na tip osnovnog objekta, najčešće poznat u drugim jezicima kao tip *Object*.

Enkapsulacija je mehanizam koji olakšava grupiranje podataka i metoda koje vrše operacije s tim podacima. Na Slici 4.4.: se vidi primjer okupljanja metode `drive` i podataka za klasu tipa `Auto`.



Sl. 4.4.: Primjer enkapsulacije u TypeScriptu (lijevo) i Javascriptu (desno)

4.3. Usporedba TypeScripta i Javascripta

TypeScript ima jedinstvenu filozofiju u usporedbi s drugim jezicima koji se nastavljaju na Javascript. Javascript program je točan TypeScript program; TypeScript je nadskup (engl. *superset*) Javascriptu. Moguće je preimenovati .js datoteke u .ts datoteke i početi koristiti TypeScript. TypeScript datoteke se prevode u čitljivi Javascript, tako da je migracija laka, a razumijevanje prevedenog TypeScripta uopće nije teško. Na taj način TypeScript se temelji na uspjesima Javascripta, a poboljšava svoje slabosti [10].

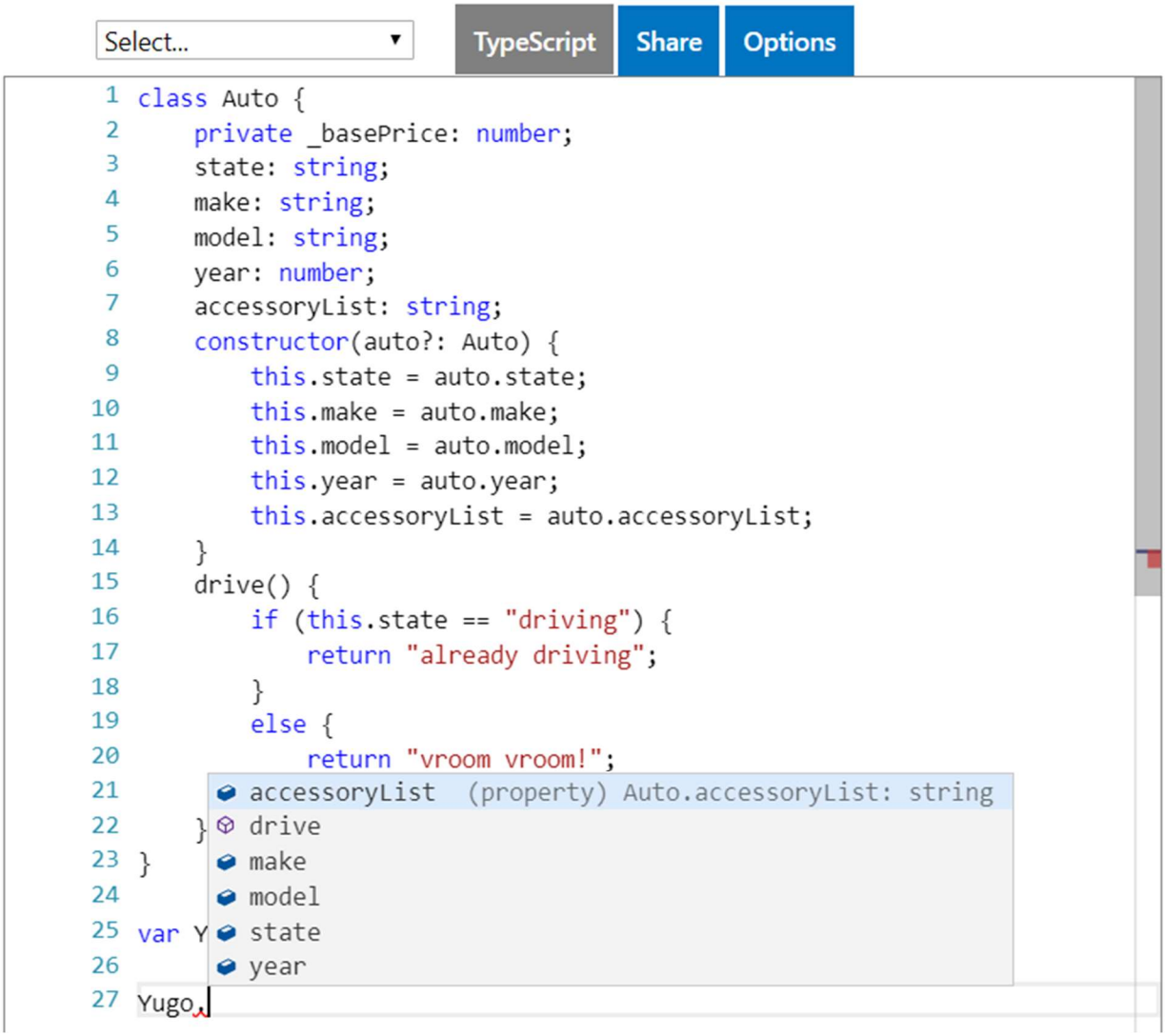
S jedne strane, to je alat koji će u budućnosti nastaviti uzimati moderne ECMAScript standarde i prenositi ih na starije Javascript inačice s kojima je Babel najpopularniji. S druge strane, postoje jezici koji se mogu potpuno razlikovati od Javascripta koji ciljaju Javascript, kao što su CoffeeScript, Clojure, Dart, Elm, Haxe, ScalaJs. Ti jezici, premda bi mogli biti bolji od budućeg Javascripta, dolaze s većim rizikom da ne budu dovoljno prihvaćeni.

TypeScript se nalazi između ove dvije krajnosti, tako da uravnotežuje rizik i dobit. TypeScript nije rizičan izbor bilo kojeg standarda. TypeScript je vrlo jednostavan za naučiti i vrlo je sličan Javascriptu jer nije posve drugačiji jezik, ima izvrsnu podršku za interoperabilnost Javascripta i sve više i više je usvojen.

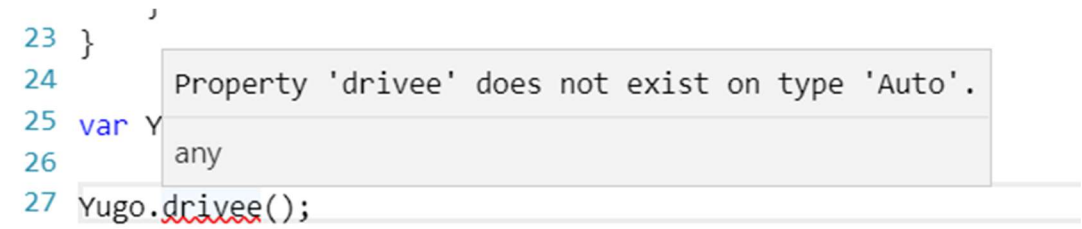
Poboljšana je i podrška za IDE[7] (engl. *Integrated Development Environment* - integrirano razvojno okruženje). Iskustvo razvoja s TypeScriptom naprednije je u odnosu na Javascript. TypeScript prevoditelj integriran je u IDE te u stvarnom vremenu obavještava o greškama u kodu. To daje nekoliko glavnih prednosti. Na primjer, pomoću TypeScripta je moguće sigurno raditi promjene (engl. *refactory*) poput preimenovanja varijabli ili funkcija kroz cijeli program. Tokom pisanja programa omogućeno je dobivanje pomoći u retku (engl. *inline*) na sve funkcije koje knjižnica(engl. *library*) može ponuditi. Pogreške u prevođenju pojavljuju se izravno u IDE-u podvlačenjem netočnog programa crvenom linijom. Sve u svemu to omogućava značajnu dobit u produktivnosti u usporedbi s radom s Javascriptom. Provodi se više vremena na pisanje programa nego na ispravljanje grešaka. Primjer vidimo na Slici 4.6..

Postoji širok raspon IDE-ova koji imaju izvrsnu podršku za TypeScript. Glavni među njima su *VisualStudio* i *VisualStudio Code*, *Atom*, *Sublime* i *IntelliJ* / *WebStorm*. [10]

[7] Program ili aplikacija u kojem se razvija proizvod pisanjem programskog koda, često nudi opciju pronalaska grešaka, prevođenje i pohranjivanje, npr. Microsoft Visual Studio, Notepad++, Atom



Sl. 4.5.: Prikaz inline pomoći tokom pisanja TypeScript koda



Sl. 4.6.: Prikaz pogreške u TypeScript kodu

Ovo su samo neke od prednosti, posebno one koje pomažu pri održavanju programa, njegovog bržeg pisanja i lakšeg snalaženja. [10]

Česte pogreške u izvršavanju običnog Javascript programa su prilikom pokušaja pristupanja svojstvu neodređenog ili nedefiniranog (engl. *undefined or null*) objekta i poziva metode umjesto svojstva. Te greške se događaju zbog pogrešaka tokom pisanja Javascript programa. TypeScript smanjuje vjerojatnost pojavljivanja takvih pogrešaka, budući da se ne može pristupiti varijabli koja čija svojstva i metode nisu poznate prevoditelju (s izuzetkom svojstava bilo koje tipirane varijable). Još uvijek je moguće pogrešno koristiti varijablu koja je postavljena na nedefiniranu vrijednost. Međutim, s 2.0 verzijom TypeScript-a ove vrste pogrešaka su eliminirane upotrebom tipova koji se ne mogu postaviti na nedefiniranu (*null*) vrijednost.

Uz omogućenu strogu nultu provjeru (engl. *strictNullChecks*) TypeScript prevoditelj neće dopustiti da se nedefinirana vrijednost dodjeli varijabli, osim ako se to ne napomene izričito.
Na primjer,

```
x: number = undefined
```

će rezultirati pogreškom u prevođenju (i prikazati se u IDE-u). To se savršeno uklapa u teoriju tipa, jer `undefined` nije `number`.
Jednom kada se zna da tip može biti postavljen na *null* ili *undefined*, TypeScript prevoditelj može odrediti je li sigurno koristiti varijablu ili ne. Jednostavan primjer je na Slici 4.7..

```
let x : number?;
if (x !== undefined) x +=1; // ovaj redak ce biti preveden jer je odradjena provjera na x
x += 1; // ovaj redak ce izbaciti pogresku tokom prevodjenja jer je x mozda nedefeniran
```

Sl. 4.7.: Primjer provjere nedefinirane vrijednosti

Bilo koja .js datoteka može se preimenovati u .ts i pokrenuti kroz TypeScript prevoditelj te bi rezultat bio sintaktički isti Javascript kod (ako je kod bio sintaktički ispravan). Čak i kada TypeScript prevoditelj dobiva pogreške u kompilaciji, ipak će se stvoriti .js datoteka. Svojstvo prihvaćanja .js datoteke kao unos omogućuje programeru brz i lagan početak rada s TypeScriptom.

Pogreške koje se pojave prilikom stvaranja TypeScript programa nisu pogreške koje će spriječiti kod od izvršavanja, kao što je slučaj u nekim drugim prevoditeljima. Prilikom pretvaranja Javascript kod u TypeScript greške su neizbježne. TypeScript provjerava čitav kod za valjanost i stoga mora imati točne informacije o svim funkcijama i varijablama koje se koriste, a posebno treba obratiti pozornost na definicije tipa. Čest je slučaj korištenja neke nejasne knjižnice za koju nisu dostupne definicije tipa ili je negdje u programu izmijenjen osnovni Javascript tip. U tom slučaju se moraju dati definicije tipa kako bi pogreške u prevođenju nestale. Ovo se najlakše rješava stvaranjem .d.ts datoteke i uključivanjem je u polje datoteka tsconfig.json, tako da je tip uvijek prepoznat. U toj datoteci se napominju bitni podatci o tipovima za koje TypeScript ne zna. [10]

5. APLIKACIJA ZA ODABIR TERMINA DEMONSTRATURE

5.1. Opis aplikacije

Aplikacija je stvorena iz potrebe za lakšim dogovorom demonstratora za odabir termina demonstrature. Naime, na određenim kolegijima, ponajviše na onim na ranijim godinama preddiplomskih studija postoji veći broj laboratorijskih grupa (oko 6 po studiju) i zbog toga i veći broj demonstratora koji trebaju biti prisutni. To može stvoriti problem prilikom dogovaranja i raspodjele termina. U početku je korištena *Google Sheets* datoteka, kojoj su pristup imali demonstratori i ondje su upisivali koji termin im odgovara. Problem kod tog sustava je bio što su prednost imali oni demonstratori koji su se prvi stigli upisati. Nakon prijedloga par demonstratora su uvedena dodatna pravila, kao npr. ograničen broj termina u jednom tjednu i da svaki demonstrator ima pravo samo na jednu grupu, osim ako nije postignut dogovor s drugim demonstratorom. Administrator, tj. asistent je imao pravo promijeniti sve podatke u tablici i brinuti o tome da je raspored pošten. Također je bio zadužen za unošenje novih termina.

Za izradu aplikacije je korišten transpiler TypeScript, te opisni jezik HTML i SCSS (engl. *SCSS - syntactically awesome cascading style sheets*). Većina logike, kontroleri te komunikacija s bazom napisana je u C#, na .Net Frameworku 4.5.2. Za izradu baze korišten je SQL (engl. *SQL - Structured Query Language*). Server, tokom razvoja, je IIS (engl. *IIS - Internet Information Services*) pokrenut na osobnom računalu.

Trenutna aplikacija se sastoji od 3 stranice - Prijava, Tablica i Postavke.
Na stranici Prijava vrši se prijava u aplikaciju sa korisničkim imenom i lozinkom koju demonstratoru dodjeljuje administrator/asistent na kolegiju. Nakon prijave, korisnik (u daljnjem tekstu demonstrator) biva preusmjeren na stranicu Tablica, gdje se nalazi sam raspored termina demonstrature. Više o tome u poglavlju 4.3. Na stranici postavke moguća je promjena lozinke i neke dodatne postavke ako se radi o demonstratoru.

5.2. Opis uloga

Aplikacija razlikuje 2 uloge - *Administrator* i *Demonstrator*. *Demonstrator* je obični korisnik koji može vidjeti tablicu i raspored samo za one kolegije kojima ima pristup. Kojim kolegijima ima pristup određuje *Administrator*. *Demonstrator* može vidjeti, pristupiti i izmijeniti samo dio podataka u tablici, a na stranici postavke ima samo opciju promjene lozinke.

Administrator ima puno više opcija. Osim što može mijenjati sve podatke vezane za demonstrature u svim tablicama (tj. kolegijima), ima opciju dodavanja novih kolegija, prilikom čega je obavezan definirati ime kolegija, na kojem je studiju, ime profesora te ime asistenta.

Administrator može dodati i novu grupu u pojedini kolegij. Naime, svaki kolegij inače dijeli studente u manje grupe (oko 20ak studenata po grupi) zbog fizičkih ograničenja laboratorija, te često postoji više grupa na svakom kolegiju iz laboratorijskih vježbi. Definiranjem grupa na kolegiju lakše se upravlja terminima. *Administrator* također može

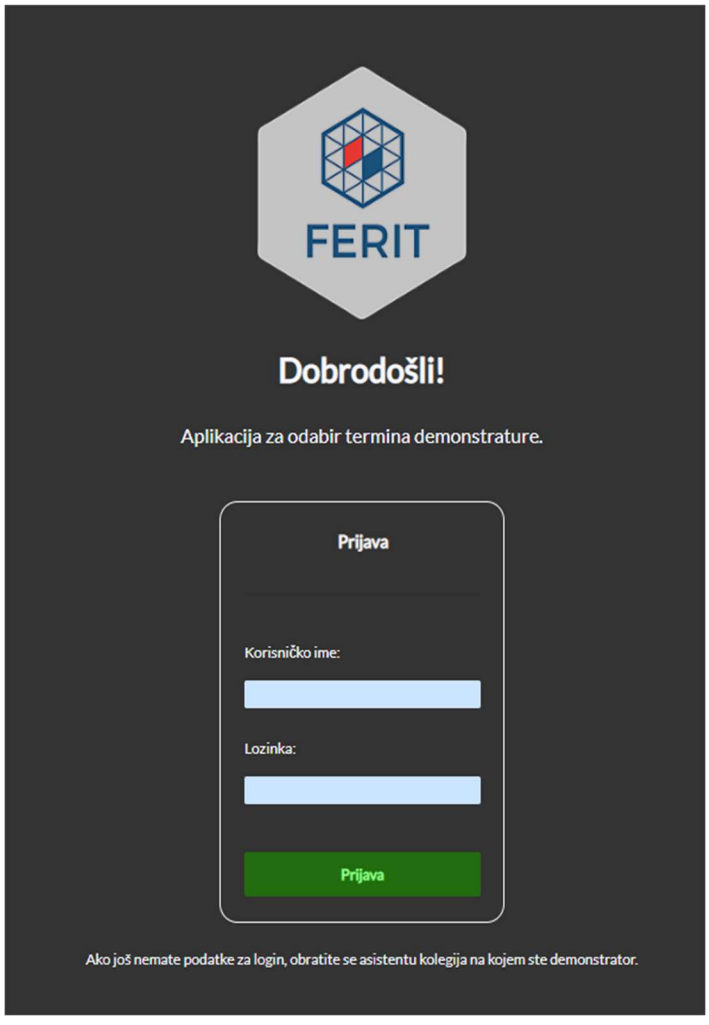
dodijeliti grupu pojedinom *demonstratoru*, što označava da taj *demonstrator* ima pravo na tu grupu i dok ne prođe određeni rok ili ne predloži svoj termin drugom *demonstratoru*, nitko ga ne može uzeti.

Administrator preko stranice Postavki također dodaje nove termine. Termin je određen datumom i grupom, te prilikom dodavanja (i brisanja) može biti dodan na sve grupe nekog kolegija ili samo na jednu grupu. Ovaj princip je napravljen najviše zbog lakšeg dodavanja jer često sve grupe imaju laboratorijske vježbe isti dan, ali to nije uvijek slučaj.

Administrator ima i opciju dodavanja korisnika, također preko stranice Postavki. Ondje unosi ime i prezime osobe, te mu dodjeljuje status administratora/demonstratora i kojim kolegijima ima pravo pristup.

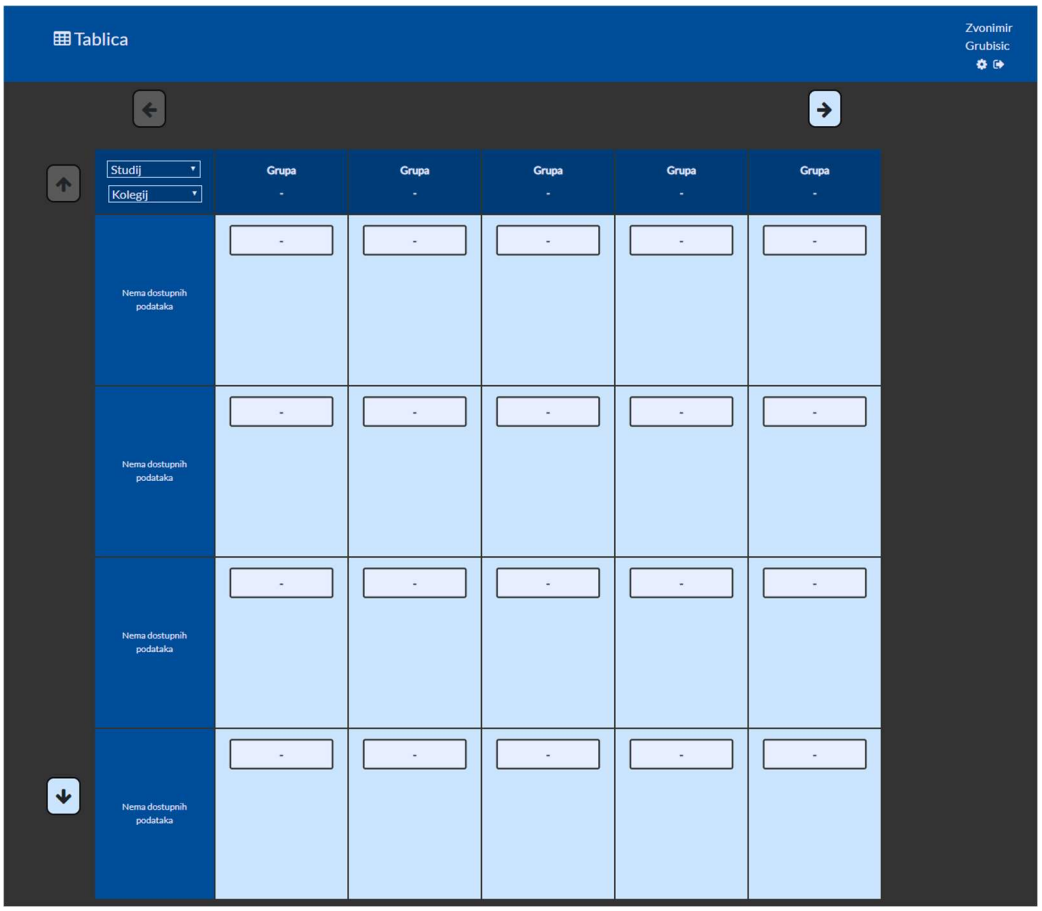
Administrator naravno može promijeniti i svoju lozinku.

5.3. Korisničko sučelje



Sl. 5.1.: Stranica za Prijavu

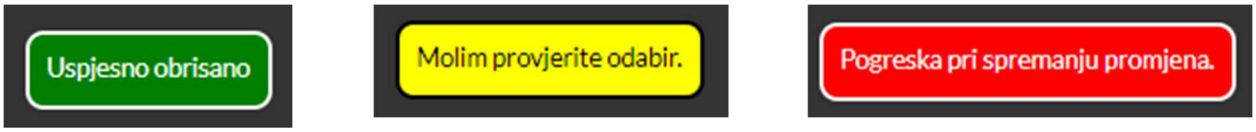
Na slici 5.1. se nalazi obrazac za prijavu - potrebno je unijeti korisničko ime i lozinku. Podatke dodjeljuje Administrator prilikom kreiranja korisnika, te ih je kasnije moguće promijeniti. Korisničko ime mora biti jedinstveno.
Polje za unos lozinke skriva tekst koji se unosi.



Sl. 5.2.: Stranica Tablica

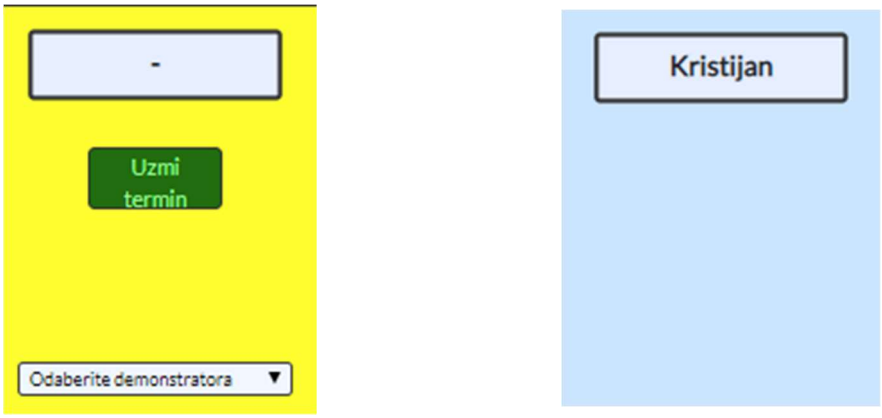
Na slici 5.2. nalazi se glavna stranica u aplikaciji.U traci na vrhu lijevo nalazi se poveznica do Tablice. Desno je ime i prezime prijavljenog korisnika, te poveznica na Postavke i Tipka za odjavu.U prvoj ćeliji se nalaze dva padajuća izbornika, za smjer na fakultetu (Elektrotehnika, Računarstvo), te ovisno o izboru smjera i kolegija dodanih za pojedini smjer.Svi korisnici imaju pravo vidjeti sve smjerove i podatke o terminima, ali mogu mijenjati samo one kojima imaju pristup.U redu na vrhu nalaze se imena grupa i ispod njih, ako im je dodijeljen, njihov vlasnik. U prvom stupcu se nalaze datumi termina. Glavni dio tablice su termini - svaka ćelija predstavlja jedan termin. Termin pripada grupi koja je iznad njega (redak na vrhu), a datum termina je vidljiv u redu tog termina u prvom stupcu.

Za navigaciju po tablici koriste se tipke na ekranu: strelice lijevo, desno, gore i dolje. Gore i dolje kontrolira starije/novije termine, dok se lijevo/desno pomiče po grupama na pojedinom kolegiju.



Sl. 5.3.: Neke od mogućih poruka tokom rada aplikacije

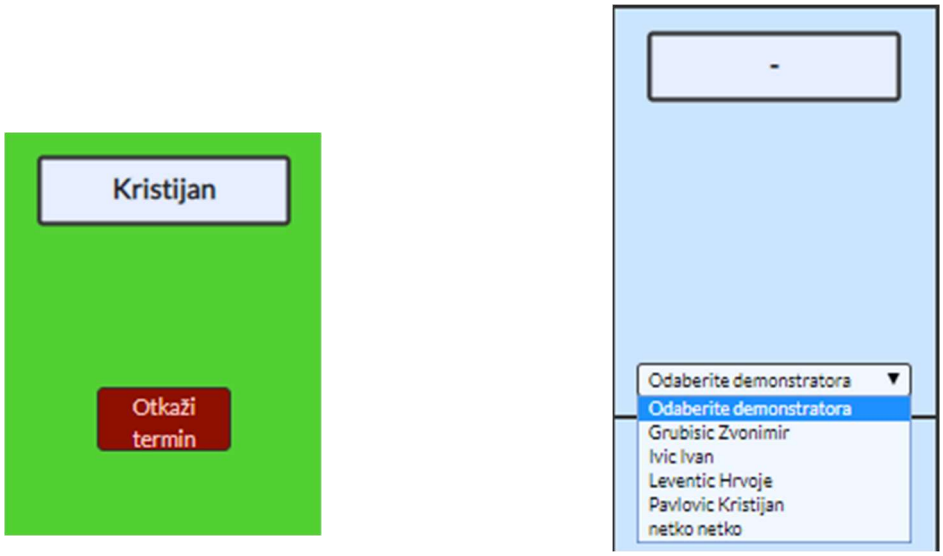
Na slici 5.3. vidljive su poruke koje se prikazuju korisniku prilikom interakcije s aplikacijom. Prikazuju se na vrhu aplikacije i "kližu" zajedno s prozorom, te nestanu nakon par sekundi.



(a) *(b)*

Sl. 5.4.: Izgled termina

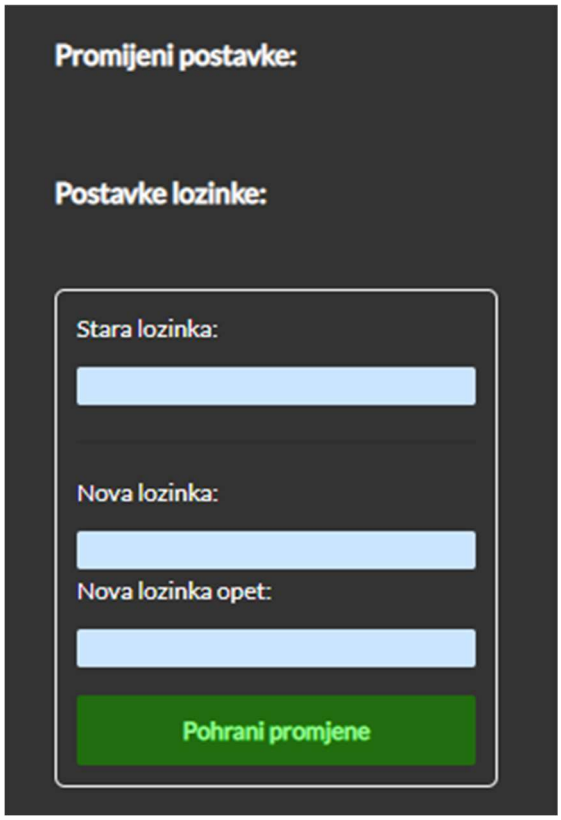
Na slici 5.4. (a) je izgled termina kada ga korisnik može rezervirati ili predložiti nekom drugom demonstratoru. (vidljiva opcija "Uzmi termin" i padajući izbornik za odabir drugog korisnika), a na slici 5.4. (b) je izgled termina kada ga zauzima drugi korisnik.



(a) *(b)*

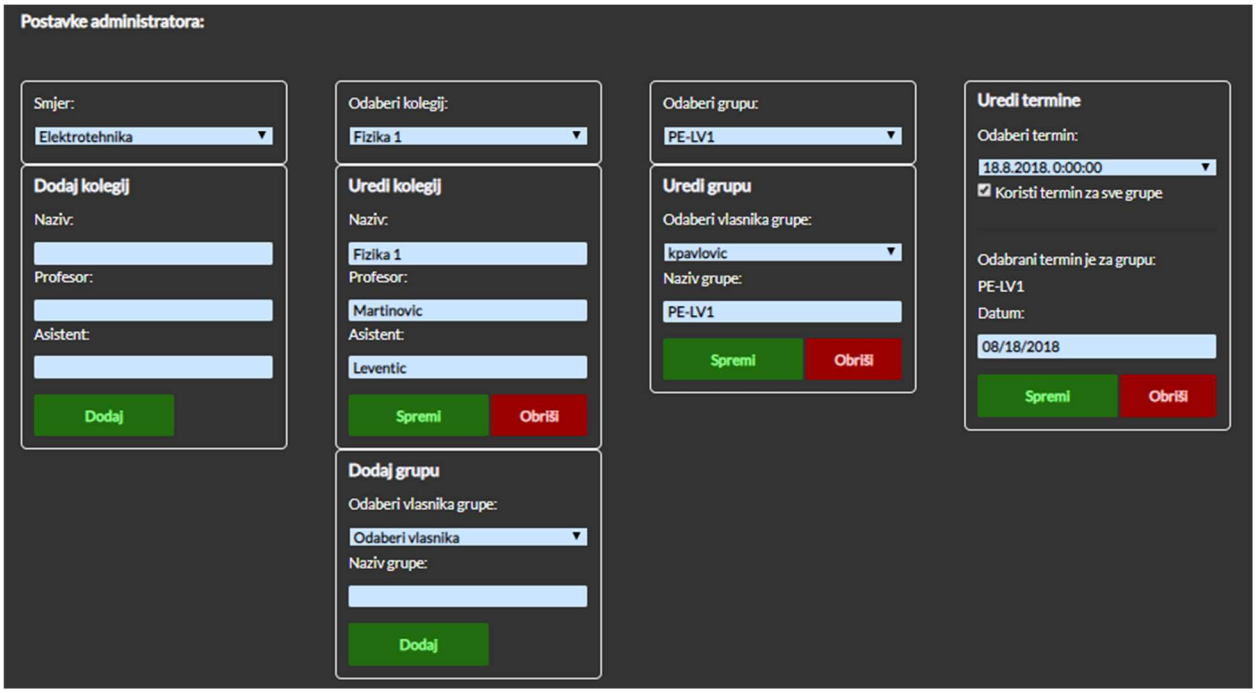
Sl. 5.5.: Izgled termina

Na slici 5.5. (a) je izgled termina kada ga je rezervirao trenutno prijavljeni korisnik.(vidljiva je opcija otkazivanja termina). Izgled termina iz perspektive administratora, dok je aktivna opcija biranja demonstratora kojem se dodjeljuje termin vidljiv je na slici 5.5 (b).



Sl. 5.6.: Stranica Postavke iz perspektive demonstratora

Klikom na kotačić u gornjem desnom kutu, korisnik dolazi na stranicu postavke. U slučaju da je prijavljenom korisniku dodijeljena uloga demonstrator, ovo su postavke koje on/ona može vidjeti: Obrazac za promjenu lozinke. Obrazac je vidljiv na slici 5.6..
Potrebno je unijeti staru (točnu) lozinku i dva puta istu (novu) lozinku te promjene spremiti klikom na “Pohrani promjene”. Nakon toga dolazi povratna obavijest o uspješnom ili neuspješnom spremanju promjena.

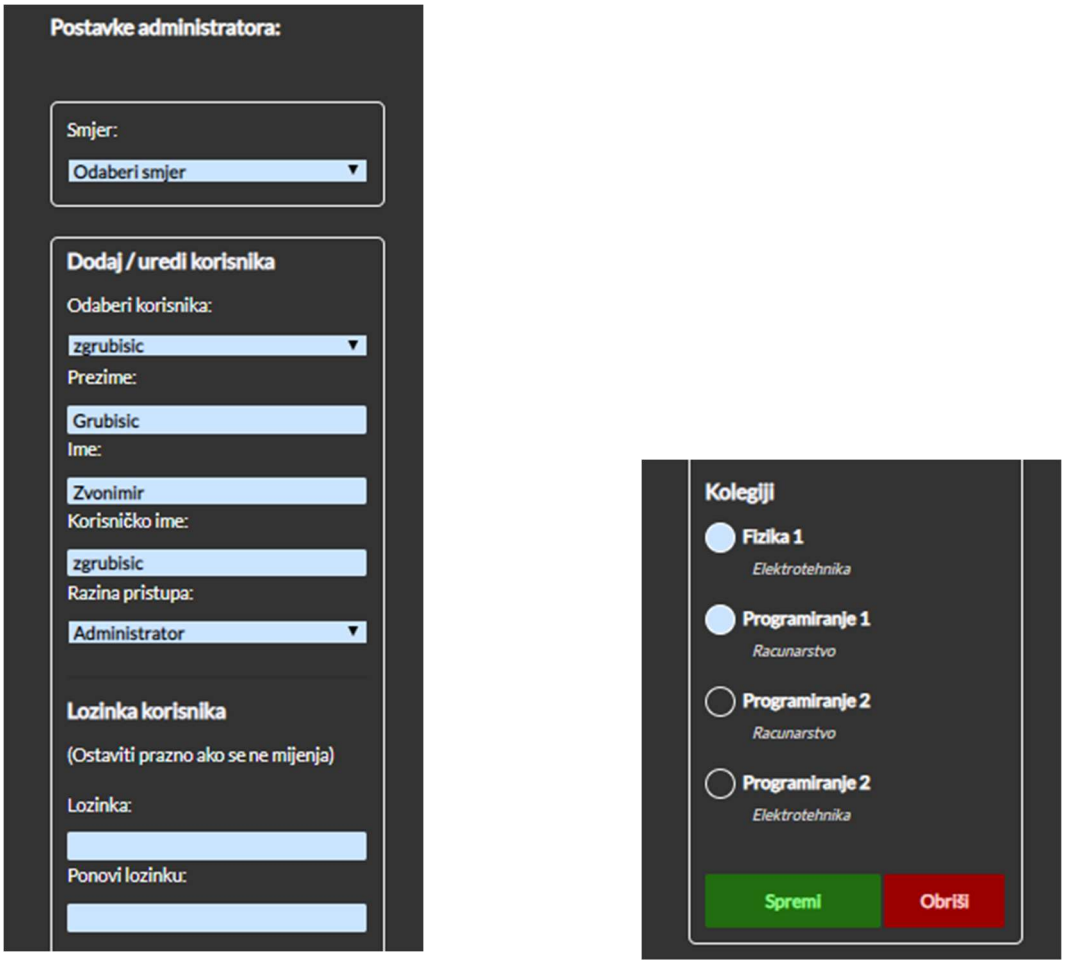


Sl. 5.7.: Stranica Postavke iz perspektive administratora

U slučaju da je prijavljenom korisniku dodijeljena uloga administrator, osim postavki za promjenu lozinke, ovo su postavke koje on/ona može vidjeti (Sl. 5.7.) :

- Dodavanje, uređivanje i brisanje kolegija (za određeni smjer)
- Dodavanje, uređivanje i brisanje grupe (za pojedini kolegij)
- Dodavanje, uređivanje i brisanje termina (za pojedinu ili sve grupe nekog kolegija)

Korisničko sučelje je osmišljeno da bude što jednostavnije i intuitivnije za korištenje.
Na vrhu se nalaze padajući izbornici za kolegij/grupu/termin.
Najsloženiji je obrazac za dodavanje termina. Prilikom odabira nekog termina, ime grupe za koju je termin napravljen je vidljivo iznad datuma termina. Također je moguće termin primijeniti za sve grupe.

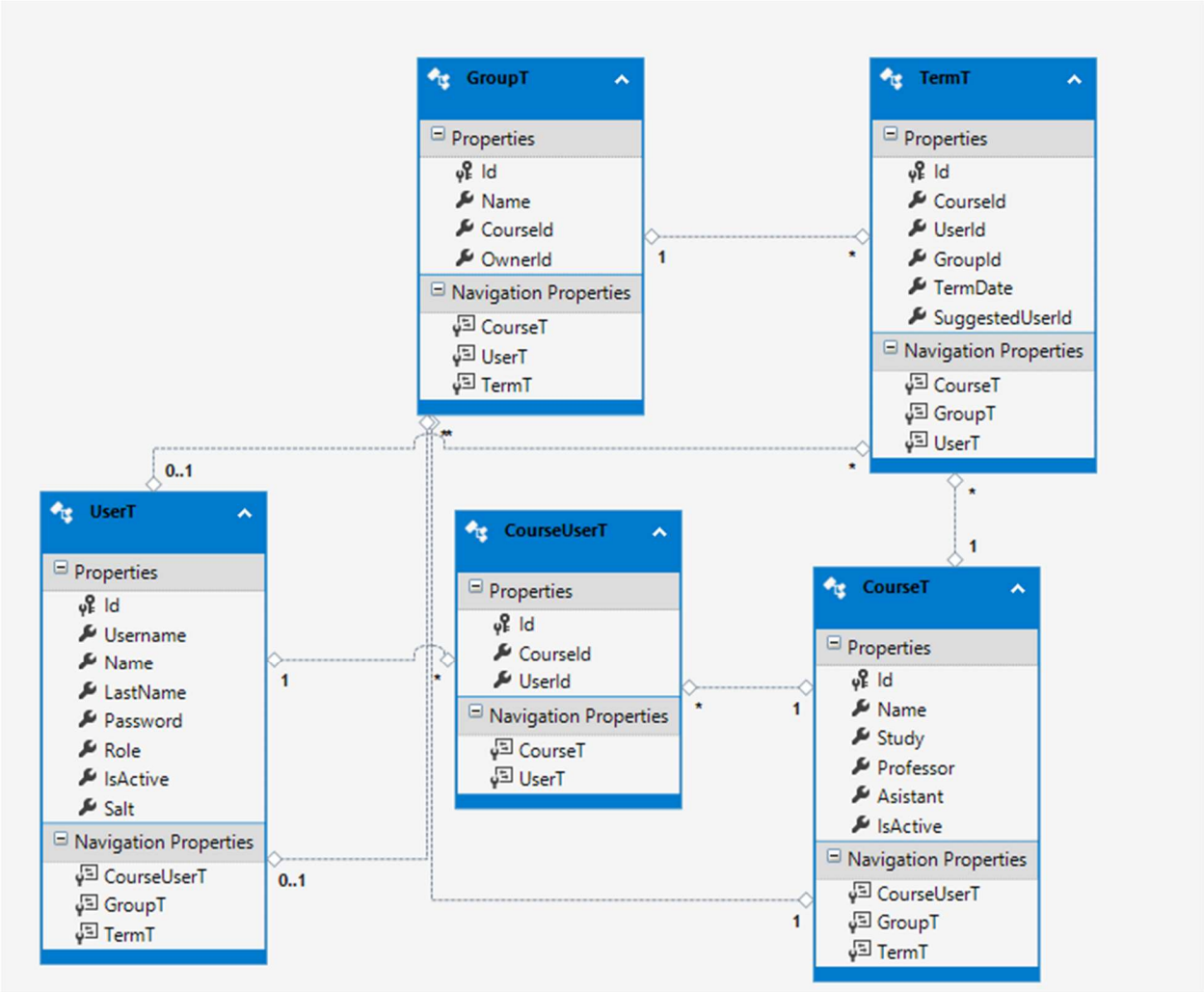


a) *b)*

Sl. 5.8.: Postavke za uređivanje korisnika

Na slikama 5.8. a) i b). su vidljive ostale postavke iz perspektive administratora. Ovdje administrator može dodati novog korisnika ili urediti postojećeg. Moguće je postaviti ime, prezime, korisničko ime te razinu pristupa. Također se jednostavno može namjestiti kojim kolegijima korisnik ima pristup. To će utjecati na izgled Tablice kada se korisnik prijavi i odabere pojedini kolegij pomoću padajućih izbornika u prvoj ćeliji tablice.

5.4. Baza podataka



Slika 5.9.: Dijagram baze podataka

Baza podataka sastoji se od 5 tablica.

- Korisnik - UserT
- Kolegij - CourseT
- KolegijKorisnik - CourseUserT
- Termin - TermT
- Grupa - GroupT

  Svojstva svake tablice vidljive su sa Slike 5.9., kao i njihovi međusobni odnosi.

Sama baza napisana je u SQL-u i napravljena je u programu Microsoft SQL Server Management Studio. Posebnu pozornost treba obratiti na tablicu UserT koja sadržava podatke o korisnicima. Polje “IsActive” se prilikom brisanja korisnika postavlja na 0 i nakon toga aplikacija više “ne zna” za tog korisnika - ali on za svaki slučaj ostaje pohranjen u bazi,
Također postoji ograničenje na Username - Korisničko ime: Nemoguće je napraviti 2 korisnika s istim imenom. U tablici se nalaze i polja Password (Lozinka) i Salt. Aplikacija koristi BCrypt sigurnosni način kriptiranja lozinke. Zahvaljujući ovome, lozinka korisnika nikad nije spremljena u bazu i nije vidljiva administratorima aplikacije.

6. ZAKLJUČAK

Cilj ovog završnog rada bio je stvoriti aplikaciju za pravljenje rasporeda demonstrature za više korisnika, s razlikovanjem uloga korisnika, njihovih prava i mogućnosti. To je bilo planirano postići kroz web aplikaciju, gdje je baza napisana u MSSQL-u, logika i komunikacija s bazom na C# .NET programskom okružju, a izgled i funkcionalnost stranice u HTMLu, SCSSu i TypeScriptu.

Rezultat je web aplikacija s responzivnim web sučeljem, koja podržava veliki broj korisnika, njihovu autentifikaciju i autorizaciju, kompatibilnost s mnogim kolegijima te dodavanje istih, te isto tako rad s velikim brojem termina i grupa za demonstrature na kolegijima. Aplikacija podržava BCrypt sigurnosno kriptiranje lozinki korisnika.

Zbog korištenja Javascript transpilera TypeScript, programiranje aplikacije je znatno olakšano, ponajviše zbog implementiranih objektno-orijentiranih svojstava u Javascriptu kroz TypeScript. Ovo također omogućava veću i bolju čitljivost koda, te lakše održavanje.

LITERATURA

[1] Flanagan, David (2011). Javascript: The Definitive Guide (6th ed.). O'Reilly & Associates. ISBN 978-0-596-80552-4,
Pristupljeno: 27.8.2018.

[2] Peleke Sengstacke, "JavaScript Transpiler: What They Are & Why We Need Them", Scotch, 2016, https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them,
Pristupljeno 27.8.2018.

[4] Charles Pick, "Why Babel Matters", codemix, 2015, http://codemix.com/blog/why-babel-matters ,
Pristupljeno 27.8.2018.

[5] "Babel Closure Elimination", https://github.com/codemix/babel-plugin-closure-elimination ,
Pristupljeno: 27.8.2018.

[6] Andrew Chalkley, "The Absolute Beginner's Guide to CoffeeScript", treehouse
http://blog.teamtreehouse.com/the-absolute-beginners-guide-to-coffeescript ,
Pristupljeno: 27.8.2018.

[7] Cliff Stanford, "Is CoffeScript still relevant or is it a has-bean?", Medium, 2016, https://medium.com/@cliff666/is-coffeescript-still-relevant-or-is-it-a-has-bean-f26c6cd9b472 ,
Pristupljeno: 27.8.2018.

[8] „What is CoffeeScript?",
https://arcturo.github.io/library/coffeescript/01_introduction.html ,
Pristupljeno: 27.8.2018.

[9] James Lavin, "The Little Book on CoffeeScript",
http://jameslavin.com/Little_Book_on_CoffeeScript.pdf ,
Pristupljeno: 27.8.2018.

[10]https://stackoverflow.com/questions/12694530/what-is-typescript-and-why-would-i-use-it-in-place-of-javascript,
Pristupljeno: 27.8.2018.

[11] „TypeScript“ http://www.typescriptlang.org/,
Pristupljeno: 27.8.2018.

[12] Charles, “Anders Hejlsberg: Introducing Typescript”, Channel 9, 2012,
https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript,
Pristupljeno: 27.8.2018.

[13] Premshree Pillai, “Introduction to Static and Dynamic Typing”, Sitepoint, 2004,
https://www.sitepoint.com/typing-versus-dynamic-typing/,
Pristupljeno: 27.8.2018.

[14] ”Osnovni koncepti i prednosti OOP u Javi”, Znanje,
http://www.znanje.org/knjige/computer/Java/ib01/300Java/31000120.htm,
Pristupljeno: 27.8.2018.

[15] ”Java Polymorphism”, TutorialsPoint,
https://www.tutorialspoint.com/java/java_polymorphism.htm,
Pristupljeno: 27.8.2018.

[16] Don Kiely, “The State of JavaScript: Language Versions”, ITProToday, 2015,
https://www.itprotoday.com/web-development/state-javascript-language-versions ,
Pristupljeno: 27.8.2018.

[17] Jen Looper, “A Guide to JavaScript Engines for Idiots”, Telerik, 2015,
https://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/ ,
Pristupljeno: 27.8.2018.

[18] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey ,
Pristupljeno: 27.8.2018.

[19] https://github.com/Microsoft/ChakraCore ,
Pristupljeno: 27.8.2018.

[20] https://rubygems.org/gems/rails/versions ,
Pristupljeno: 27.8.2018.

[21] PANU PITKÄMÄKI, “ES6 vs ES2015 - What to call a JavaScript version?”,
https://bytearcher.com/articles/es6-vs-es2015-name/,
Pristupljeno: 27.8.2018.

SAŽETAK

U ovom radu predstavljen je koncept transpilera (engl. *transpilers*) tj. *source-to-source* prevoditelja. Ono što transpileri zapravo čine je pretvaranje, tj. prevođenje jednog programskog jezika u drugi programski jezik. Ovaj rad je usredotočen na Javascript transpilere za koje kažemo da ciljaju Javascript programski jezik.

Kroz rad je predstavljeno više transpilera, a u najviše detalja je opisan Babel, CoffeeScript i TypeScript. Tome je prethodio uvod u ECMAScript i njegove verzije kroz godine.

Babel je transpiler za Javascript koji može prevesti najnoviju verziju Javascripta u kod koji se pokreće u svakom internetskom pregledniku. Opisana su njegova svojstva, dani primjeri koda i primjeri nove sintakse koja dolazi u ES17 te je prikazana ugrađena podrška za *React JSX* nadodatke te *Flow* anotacije.

Opisan je i CoffeeScript transpiler, transpiler koji omoguće korištenje mnogih značajki Javascripta dodavanjem sintaktičkog šećera (engl. *syntactic sugar*) u Javascript. Prikazana i opisana je sintaksa CoffeeScripta, koja nema točke zareze i vitičaste zagrade, te ima sličnu sintaksu Pythonu i Rubyju. Iz ovih razloga je pisanje CoffeeScript koda brzo i bezbolno, te ista svojstva olakšavaju čitanje i održavanje. Opisane su i neke ključne riječi CoffeeScripta, način kako CoffeeScript radi s određenim operatorima i integracija strogog načina. Može se zaključiti da CoffeeScript kod nakon prevođenja postaje jednako uspješan ili u nekim slučajevima još učinkovitiji Javascript kod.

Naposljetku je opisan i TypeScript transpiler. Navede su bitne značajke TypeScripta, a one su neobavezno statični tipovi u kodu, te koncept i realizacija klasa i sučelja. Najveća prednost TypeScripta, osim što u Javascript programski jezik uvodi objektno orijentirano programiranje, što pruža mogućnost bogatijeg razvojnog okruženja gdje je puno lakše uočiti precizne greške prilikom pisanja koda. Pisanje programa u TypeScriptu je brzo i zbog dobrog prevoditelja, broj grešaka je minimalan.

ABSTRACT

This paper introduced the concept of transpilers (source-to-source translators). What transpilers actually do is converting, i.e. translating a programming language into another programming language. This paper focuses on the Javascript transpilers which target Javascript programming language.

Several transpilers have been introduced through the paper with most focus on Babel, CoffeeScript and TypeScript. Before that necessary introduction to ECMAScript and it's versions throughout years was presented.

Babel is a Javascript transpiler that can translate the latest version of Javascript into the code that is run on every internet browser. This paper described its features, presented code examples, examples of new syntax coming to ES17 and built-in features necessary for support React JSX additions and Flow Annotation.

Papers also described CoffeeScript transpiler, a transpiler that enables the use of many Javascript features by adding syntactic sugar to Javascript. The CoffeeScripta syntax, which has no semicolons and curly braces, is depicted and described. CoffeeScript has similar syntax to Python and Ruby. For these reasons, writing of the CoffeeScript code is quick and painless, and it’s also easy to read and maintain. Some CoffeeScripta keywords are described, how CoffeeScript works with specific operators and strict mode. It can be concluded that the CoffeeScript code after translating becomes equally successful or in some cases even more effective Javascript code.

Finally, TypeScript transpiler is also described. Presented are the essential features of TypeScript, which are optional static types in the code, as well as concept and realization of classes and interfaces. The biggest advantage of TypeScript, beside the introduction of object-oriented language features in Javascript is the fact it provides a richer development environment where it is much easier to perceive precise errors when writing code. Writing a program in TypeScript is fast and due to its great compiler, the number of errors remains at minimum.

ŽIVOTOPIS

Zvonimir Grubišić rođen je 26. veljače 1995. u Osijeku u Hrvatskoj. 2002. upisuje Osnovnu školu Frana Krsta Frankopana u Osijeku. Nakon završetka osnove škole, 2010. godine upisuje Prirodoslovno-matematičku gimnaziju u Osijeku koju završava 2014. godine i ostvaruje izravan upis na preddiplomski studij Računarstvo na Elektrotehničkom fakultetu Osijek. Pokraj studija, aktivan je član studentske udruge IAESTE, pohađa stručne prakse u Košicama u Slovačkoj i u Oslu u Norveškoj, te godinu dana radi u tvrtki GDi.

(Zvonimir Grubišić)

PRILOZI

Prilog 2.1. - Tablica o razlikama ECMAScript 2016+

ECMAScript 5 6 2016+ next intl non-standard compatibility table
Flattr
by kangax & webbedspace & zloirock
Fork 669
Sort by Features
Show obsolete platforms
Show unstable platforms
V8 SpiderMonkey JavaScriptCore Chakra Other
Minor difference (1 point) Small feature (2 points) Medium feature (4 points) Large feature (8 points)
Feature name
Current browser
CH 69, OP 56
CH 68, OP 55
GraalVM 1.0[3]
iOS 11.3
SF 11.1
FF 62
FF 60 ESR
FF 61
Node >=8.10 <9[2]
iOS 11
SF 11
Edge 16
Babel 6 + core-js
Type-Script + core-js
Edge 17
Closure 2018.08
Node >=6.5 <7[2]
es7-shim
DUK 2.2
Traceur
Node 4[2]
PJS
IE 11
2016 features
exponentiation (**) operator
Array.prototype.includes
2016 misc
generator functions can't be used with "new"
generator throw() caught by inner generator
strict fn w/ non-strict non-simple params is error
nested rest destructuring, declarations
nested rest destructuring, parameters
Proxy, "enumerate" handler removed
Proxy internal calls, Array.prototype.includes
2017 features
Object static methods
String padding
trailing commas in function syntax
async functions
shared memory and atomics
2017 misc
Proxy "ownKeys" handler, duplicate keys for non-extensible targets (ES 2017 semantics)
RegExp "u" flag, case folding
arguments.caller removed

2017 annex b																									
Object.prototype getter/setter methods	▶	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	14/16	16/16	16/16	14/16	0/16	10/16	0/16	16/16	0/16	10/16	12/16	8/16
Proxy internal calls, getter/setter methods	▶	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
assignments allowed in for-in head in non-strict mode	C	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes
2018 features																									
object rest/spread properties	▶	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	2/2	2/2	0/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
Promise.prototype.finally	▶	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	0/3	0/3	0/3	3/3	3/3	0/3	3/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3
s (dotAll) flag for regular expressions	C	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	No	No	No	Yes	?	No	No	?	No	No	?	?	No	No	
RegExp named capture groups	C	Yes	Yes	Yes	Yes	Yes	No	No	No	Flag[23]	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
RegExp Lookbehind Assertions	C	Yes	Yes	Yes	No	No	No	No	No	Yes	No	No	No	No	No	No	No	Flag[23]	No	No	No	No	No	No	No
RegExp Unicode Property Escapes	C	Yes	Yes	Yes	Yes	Yes	No	No	No	Flag[23]	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No
Asynchronous Iterators	▶	2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	2/2	0/2	0/2	0/2	2/2	2/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
2018 misc																									
template literal revision	C	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	No	No
2019 misc																									
optional catch binding	▶	3/3	3/3	3/3	0/3	3/3	3/3	3/3	3/3	3/3	0/3	0/3	0/3	0/3	0/3	3/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3	0/3

[1] Flagged features have to be enabled via `--harmony` flag

[2] Flagged features have to be enabled via `--harmony` or `--es_staging` flag

[3] Executed in Node.js/JVM mode via `graalvm/bin/node --jvm`.

[4] The feature have to be enabled via "Experimental Javascript features" setting under `about:flags`

[5] This feature is supported when using Babel with core-js.

[6] This feature is supported when using TypeScript with core-js.

[7] TC39 meeting notes from July 28, 2015.

[8] 'Semantics of yield* in throw case' GitHub issue in ECMA-262 repo.

[9] Requires the `downlevelIteration` compile option.

[10] TC39 meeting notes from July 29, 2015.

[11] TC39 meeting notes from July 28, 2015.

[12] TC39 meeting notes from July 28, 2015.

[13] 'Normative: Remove [[Enumerate]] and associated reflective capabilities' GitHub Pull Request in ECMA-262 repo.

[14] Flagged features have to be enabled via "Enable experimental Javascript features" setting under `about:flags`

[15] This feature requires native generators or `regenerator-runtime`, it's a part of `babel-polyfill` or `babel-runtime`.

[16] The feature was temporarily disabled to mitigate the Meltdown and Spectre CPU bugs.

[17] The feature is available only in Firefox Developer Edition and Firefox Nightly builds.

[18] The feature was temporarily disabled to mitigate the Meltdown and Spectre CPU bugs.

[19] The feature was temporarily disabled to mitigate the Meltdown and Spectre CPU bugs.

[20] The feature was temporarily disabled to mitigate the Meltdown and Spectre CPU bugs.

[21] The feature have to be enabled via "Experimental enabled SharedArrayBuffer support in JavaScript." setting under `about:flags`

[22] The behaviour of the Proxy "ownKeys" handler in presence of duplicate keys has been modified later.

[23] The feature have to be enabled via `--js-flags="--harmony"` flag

[24] The feature is considered unstable, but can be enabled via `--js-flags="--harmony-promise-finally"` flag

[25] The feature is available only in Firefox Nightly builds.