Grafičko okruženje za generiranje automatskih testova i dokumentacije za ADAS sustave

Birtić, Dominik

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

Sveučilišni studij

Grafičko okruženje za generiranje automatskih testova i dokumentacije za ADAS sustave

Diplomski rad

Dominik Birtić

Osijek, 2019.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 06.09.2019.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Dominik Birtić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-927R, 19.09.2018.
OIB studenta:	30797973126
Mentor:	Izv. prof. dr. sc. Mario Vranješ
Sumentor:	
Sumentor iz tvrtke:	Marko Halak
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Marijan Herceg
Član Povjerenstva:	Doc.dr.sc. Ratko Grbić
Naslov diplomskog rada:	Grafičko okruženje za generiranje automatskih testova i dokumentacije za ADAS sustave
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Budući da su ADAS (engl. Advanced driver-assistance systems) sustavi često vrlo složeni, njihova programska podrška zahtjeva temeljito testiranje prije upotrebe, što podrazumijeva pisanje i izvršavanje velikog broja testova. U okviru diplomskog rada potrebno je osmisliti grafičko okruženje koje bi olakšalo stvaranje testova za ADAS sustave. Predloženo rješenje treba omogućiti korisniku odabir preduvjeta i željenih akcija u testovima, slaganje njihovog redoslijeda i definiranje očekivanih rezultata, a kao rezultat okruženje generira Python skriptu i tekstualni opis svih koraka koji su odabrani. Automatska skripta bi se sastojala od poziva već postojećih funkcija iz odgovarajuće biblioteke. Osim toga, okruženje treba omogućiti učitavanje već ranije napravljenih testova, kako bi se olakšale eventualne ispravke na njima. Samo okruženje mora biti fleksibilno u što je moguće većoj mjeri, kako bi omogućilo sastavljanje različitih testova. Kao osnova za grafičko sučelje može se koristiti neko od postojećih rješenja.
Prijedlog ocjene pismenog dijela ispita (diplomskog 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:	06.09.2019.

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, 18.09.2019.

Ime i prezime studenta:	Dominik Birtić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-927R, 19.09.2018.
Ephorus podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Grafičko okruženje za generiranje automatskih testova i dokumentacije za ADAS sustave**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Mario Vranješ

i sumentora

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:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

Ja, Dominik Birtić, OIB: 30797973126, student/ica na studiju: Diplomski sveučilišni studij Računarstvo, dajem suglasnost Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek da pohrani i javno objavi moj **diplomski rad**:

Grafičko okruženje za generiranje automatskih testova i dokumentacije za ADAS sustave

u javno dostupnom fakultetskom, sveučilišnom i nacionalnom repozitoriju.

Osijek, 18.09.2019.

_____

potpis

SADRŽAJ

1. UVOD

Zahvaljujući napretku digitalnih tehnologija, poput robotike, umjetne inteligencije i računala visokih performansi, sve više i više su se počele koristiti te tehnologije kako bi unaprijedile automobile. Autonomna vozila su automobili koji su opremljeni senzorima, kamerama, računalima, GPS-om, satelitskim prijemnicima, radarima kratkog dometa i LIDAR-om te koriste sve navedeno kako bi sve ili dio zadataka u vožnji obavili. Autonomna vozila se sve više i više pojavljuju danas, te se razina njihove autonomije povećava iz dana u dan. Napredni pomoćni sustavi za vožnju se još nazivaju ADAS. ADAS su namijenjeni da pomognu vozačima pri vožnji i da povećaju sigurnost vozila i sigurnost prometa. Danas se veliki broj inženjera bavi izradom algoritama za te sustave. Dnevno nastaje velika količina programskog koda za te sustave kojeg je potrebno testirati zbog čega je ručno testiranje i pisanje koda dugačak i naporan posao.

Budući da su ADAS (engl. Advanced driver-assistance system) često vrlo složeni, njihova programska podrška zahtjeva temeljito testiranje prije upotrebe što podrazumijeva pisanje i izvršavanje velikog broja testova. U okviru diplomskog rada koristi se grafičko okruženje nazvano *Blockly*, koje pomoću blokova koji predstavljaju programski kod, olakšava stvaranje testova za ADAS. Blokovi u *Blockly* okruženju su napravljeni na način da korisnicima slikovito prikažu dio programskog koda za koji je blok namijenjen. Blokovi sadrže kratki opis koji navodi korisnika na programski kod koji blok predstavlja. Zadatak diplomskog rada bio je napraviti okruženje koje se može povezati s bazom podataka i upotrijebiti *Blockly* kako bi se omogućilo generiranje skripti u *Python* programskom jeziku. Omogućiti korisniku jednostavno unošenje vlastitog koda napisanog u programskom jeziku *Python* i pretvoriti taj kod u blok koji se zapisuje u bazu podataka koji se kasnije u kombinaciji s drugim blokovima sprema kao *Python* skripta koja bi predstavila jednu inačicu testa za ADAS. Zahtjevi koji su trebali biti ispunjeni konačnim rješenjem su:

- Mogućnost spuštanja ili preuzimanja (engl. *download*) tekstualne datoteke i skripte napisane u *Python* programskom jeziku na osobno računalo;
- Mogućnost zapisa blokova u bazu podataka, tj. zapis koda koji predstavlja izgled i funkcionalnost blokova u bazu podataka i učitavanje blokova iz baze podataka;
- Dinamičko zapisivanje kategorije i naziva bloka u alatnu kutiju (engl. *toolbox*) u vlastitu *HTML* datoteku;

- Integracija tvornice blokova (engl. *Block Factory*) u aplikaciju, tj. integracija *Blockly* sučelja za stvaranje blokova u vlastitu web aplikaciju;
- Mogućnost unosa *Python* koda i raščlanjivanje (engl. *parsing*) varijabli s oznakom te zamjenjivanje tih varijabli s varijablama koje blok koristi;
- Stvoriti gumb koji koristi raščlanjeni (engl. *parsed*) kod i povlači *Blockly* kod koji predstavlja izgled i funkcionalnost bloka te ubacuje raščlanjeni kod u funkcionalni dio bloka;

Diplomski rad je podijeljen u pet poglavlja. Drugo poglavlje opisuje tematiku diplomskog rada i daje pregled postojećeg (zatečenog) rješenja za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju te iznosi nedostatke koji u njemu postoje i koje je potrebno riješiti u sklopu ovog diplomskog rada. Treće poglavlje opisuje vlastito rješenje, tj. rješenje koje je napravljeno u sklopu ovog rada na temelju nedostataka koji postoje u postojećem rješenju. U trećem poglavlju su također opisane korištene tehnologije za izradu vlastitog rješenja. U četvrtom poglavlju dani su rezultati verifikacije stvorenog rješenja, tj. opisan je način kako su se provjeravale funkcionalnosti koje je trebalo implementirati s obzirom na početne zahtjeve. U petom poglavlju izneseni su zaključci diplomskog rada.

2. PROBLEM GENERIRANJA SKRIPTI ZA TESTIRANJE NAPREDNIH SUSTAVA ZA POMOĆ VOZAČU U VOŽNJI

Ovo poglavlje opisuje pobliže tematiku kojom se bavi sami diplomski rad, a to je problematika generiranja skripti za automatsko testiranje naprednih sustava za pomoć vozaču u vožnji (ADAS). Opisana je i vrlo korisno *Blockly* okruženje kao i jedno postojeće (zatečeno) rješenje za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju. Dan je pregled nedostataka koji postoje u tom rješenju i opisano je kako ti nedostaci utječu na izvedbu posla kojeg postojeće rješenje treba obavljati.

2.1 Testiranje naprednih sustava za pomoć vozaču

Ovaj diplomski rad bavi se temama automatskog testiranja ADAS, stvaranja testova i okruženja u kojem se mogu stvarati automatski testovi. ADAS pružaju novu razinu sigurnosti vozačima i vozilima i na taj način mogu spasiti brojne živote. Teme automatskog testiranja spominju se u [1, 2]. Kao što je rečeno u prvom poglavlju, ADAS zahtijevaju veliku količinu testiranja kako bi se provjerili razni slučajevi na koje ADAS može naići prilikom svog rada. To testiranje se može izvršiti automatski, čime se štedi velika količina vremena inženjera, a koje bi se inače potrošilo na ručnu izradu i provjeru testova. Tržište ADAS je trenutno ograničeno zbog nedovoljne pouzdanosti rada samog softvera, kao i cijene takvih sustava za široku namjenu [3]. No u budućnosti je cilj podići razinu pouzdanosti njihova rada te sniziti cijenu koštanja.

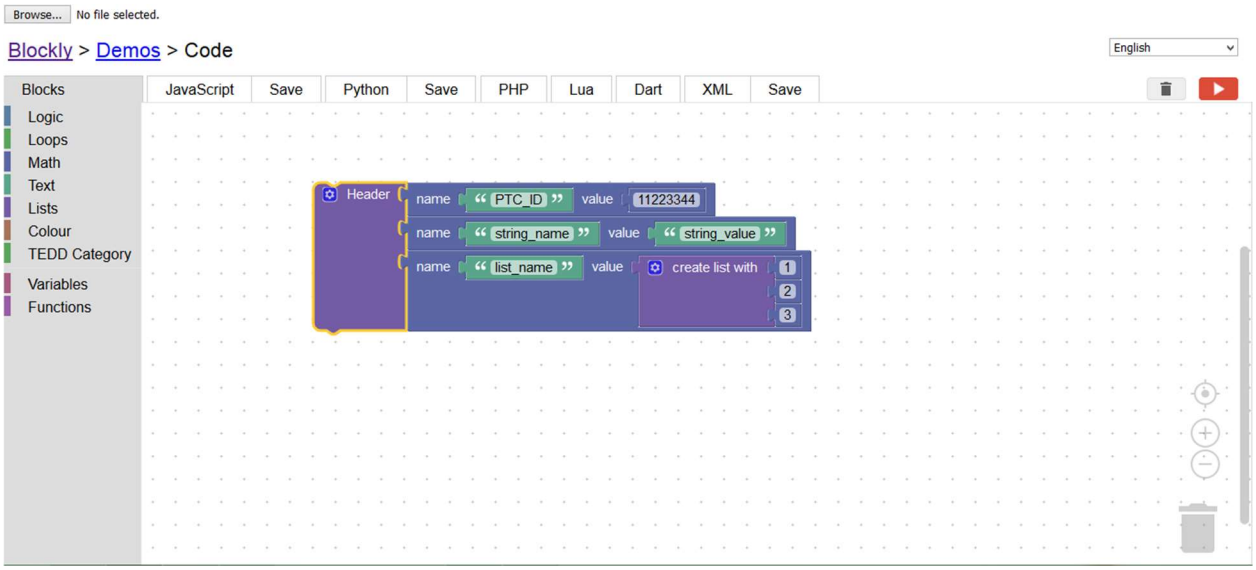
2.2 Blockly okruženje

Blockly je okruženje koje dodaje vizualni uređivač programskog koda web i mobilnim aplikacijama [4]. Njegov uređivač koda koristi takozvane blokove koji predstavljaju programski kod. Blokovi se povezuju poput slagalice kojoj je krajnji cilj sastaviti kod koji će predstavljati funkcionalnu skriptu koju bi krajnji korisnik Blockly-ja mogao koristiti za učenje, testiranje, za isprobavanje različitih koncepata programiranja npr. grananje, petlje i slično. Blockly je nastao kako bi na jednostavan i zanimljiv način naučio učenike ili studente [4] osnovne koncepte programiranja, bez da se obaziru na sintaksu pisanja koda te uvodi koncepte programiranja kao što su petlje, varijable, grananje, logički izrazi i slično. Blockly je napisan u JavaScript jeziku, no njegovi blokovi mogu predstavljati i kodove drugih programskih jezika kao na primjer Python, PHP, Lua i slično. Iz perspektive korisnika, Blockly je intuitivan i vizualan način za građenje

programskog koda. Iz perspektive programera, Blockly je gotovo grafičko okruženje za stvaranje vizualnog jezika koje emitira sintaktički ispravan korisnički generirani kod [4].

2.3 Postojeće rješenje za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju

Postojeće rješenje za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju je web stranica koja koristi *Blockly* okruženje. *Blockly* okruženje omogućuje korisnicima da naprave skripte u različitim programskim jezicima koristeći blokove. Svaki blok predstavlja jednu ili više linija programskog koda. Namjena postojećeg rješenja je stvoriti skripte u *Python* programskom jeziku koje će moći izvesti jedan test za ADAS. Postojeće stanje rješenja za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju omogućuje slaganje proizvoljnih blokova i može prikazati kod tih blokova napisanog u *JavaScript* i *Python* programskim jezicima. Postojeće rješenje također omogućuje mogućnost preuzimanja tog koda u obliku skripte napisane u *JavaScript*, *Python* i *XML* jeziku te omogućuje pokretanje tog programskog koda. Ovo rješenje također omogućuje prikaz koda u jezicima *PHP*, *Lua*, *Dart* i *XML*, no prikaz koda u tim jezicima ne radi za sve blokove koji se mogu koristiti. Na slici 2.1 je prikazano grafičko sučelje postojećeg rješenja.



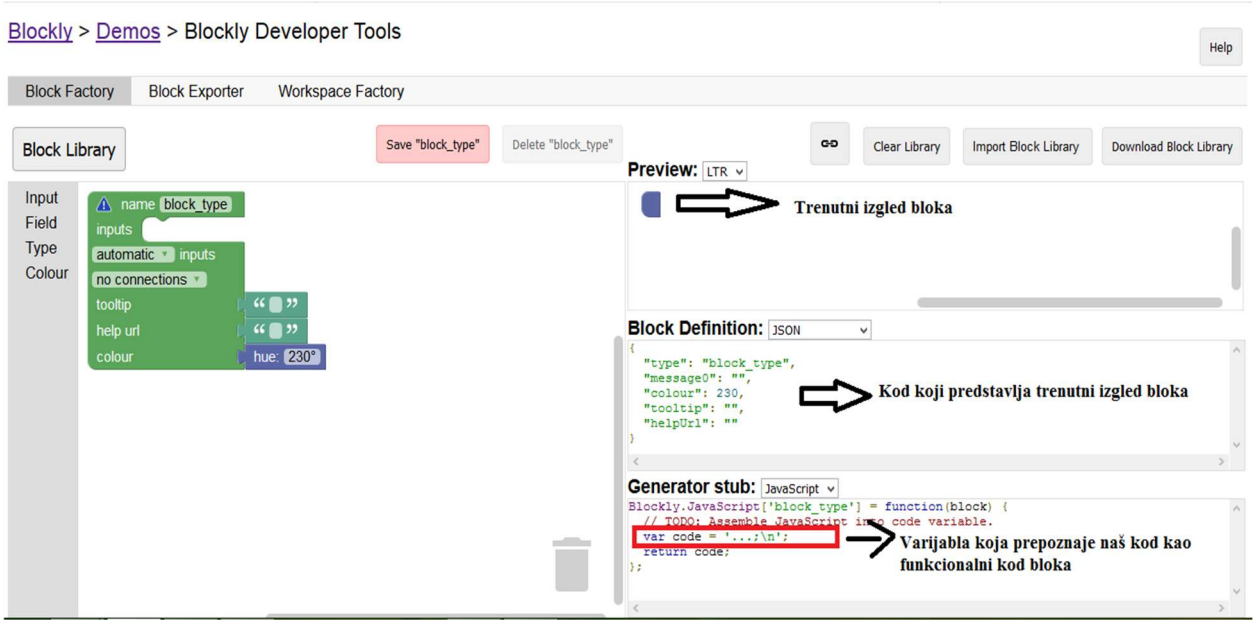
Slika 2.1. Prikaz grafičkog sučelja postojećeg rješenja

Kao što je na slici 2.1 prikazano, postojeće rješenje koristi *Blockly* okruženje za slaganje blokova i koristi *Blockly* prikazivač koda pomoću kojeg se programski kod može prikazati u

različitim jezicima, kao što je prethodno bilo navedeno. Standardno *Blockly* okruženje sastoji se od alatne kutije i radnog prostora (engl. *workspace*), te se u radnoj kutiji nalaze blokovi koji se mogu koristiti.

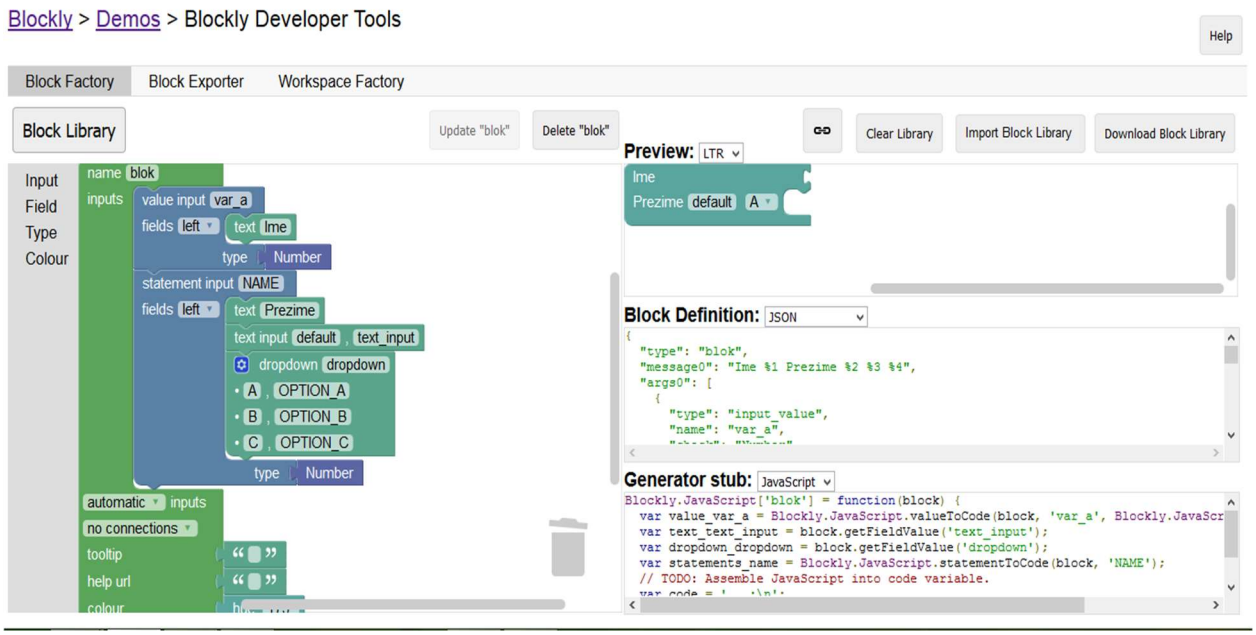
Inače se blokovi u *Blockly* okruženju prave korištenjem *Blockly Developer Tools (engl. Blockly Developer Tools)* lokalno ili u pregledniku. Kada se napravi vlastiti blok, treba preuzeti kod tog bloka ili ga označiti i kopirati u lokalnu *JavaScript* datoteku te jedan dio koda treba upotpuniti s vlastitim kodom kojeg će taj blok izvršavati. Nakon što je kod za funkcionalnost bloka dovršen i *JavaScript* datoteka spremljena, tada se mora zapisati blok u alatnu kutiju u *HTML* datoteci. Budući da postojeće rješenje ne sadrži opisane funkcionalnosti, onda se manjak te funkcionalnosti vidi kao prvi nedostatak.

Na slici 2.2 prikazani su razvojni alati koji se koriste za generiranje vlastitih blokova. Označeni su i kratko opisani dijelovi razvojnih alata na koje treba obratiti pažnju, kao što su *Block Definition* i *Generator Stub*.



Slika 2.2. Prikaz *Blockly* razvojnih alata

Na slici 2.3 su prikazane mogućnosti i detalji kojima korisnik upravlja prilikom izrade bloka te u usporedbi sa slikom 2.2 može se vidjeti kako se izgled bloka i kod u *Block Definition* i *Generator Stub* sam dodaje i ažurira ovisno koje stavke korisnik stavi u svoj blok. Nakon što je blok napravljen i preuzet korisnik može dodati još neke dijelove koda te trebao bi dodati u varijablu *code* vlastiti kod koji će upravljati blokom.



Slika 2.3. Prikaz *Blockly* razvojnih alata – izrada bloka

Nakon što korisnik postojećeg rješenja napravi više od desetak blokova koje spremi u *JavaScript* datoteku, ta *JavaScript* datoteka postane pretrpana kodom koji je nepregledan (npr. za desetak blokova datoteka može biti i šestotinjak linija koda ili više). Ovo se vidi kao drugi nedostatak koji utječe na brzinu izmjene postojećih blokova, no utječe i na brzinu stavljanja novog bloka.

Nakon provedene analize načina rada postojećeg rješenja za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju i otkrivanja njegovih nedostataka, može se započeti izrada naprednije inačice rješenja za istu namjenu, što je glavni cilj ovog rada. Naprednija inačica rješenja je predstavljena u sljedećem poglavlju, a bitno je za napomenuti da se radi o potpuno novom rješenju, a ne o nadogradnji postojećeg rješenja.

3. IZRADA NOVOG RJEŠENJA ZA GENERIRANJE SKRIPTI ZA TESTIRANJE ADAS ZASNOVANOG NA *BLOCKLY* OKRUŽENJU

Na osnovu nedostataka koji su prisutni prilikom korištenja postojećeg rješenja za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju (koje je opisano u podpoglavlju 2.3), bilo je potrebno izraditi potpuno novo rješenje koje će zadovoljiti sljedeće zahtjeve:

1) potrebno je napraviti gumbove za preuzimanje *Blockly* radnog prostora kao *Python* skripte i kao tekstualne datoteke;
2) potrebno je povezati *Blockly* okruženje s bazom podataka u koju će se moći spremati blokovi;
3) potrebno je moći dinamički zapisati kategoriju i naziv bloka u alatnu kutiju u vlastitoj *HTML* datoteci;
4) potrebno je integrirati tvornicu blokova u okruženje;
5) potrebno je omogućiti unos *Python* koda u obrazac u tvornici blokova i raščlaniti varijable s oznakom i zamijeniti te varijable s varijablama koje blok koristi;
6) potrebno je stvoriti gumb koji koristi raščlanjeni kod i povlači kod koji predstavlja izgled i funkcionalnost bloka te ubacuje raščlanjeni kod u funkcionalni dio bloka;

3.1 Alati i tehnologije korištene za izradu dodatnih funkcionalnosti

3.1.1 Blockly

Budući da je *Blockly* opisan u drugom poglavlju, ovdje je opisano za što se koristi u novom vlastitom rješenju. U kontekstu diplomskog rada *Blockly* se koristi za slaganje blokova koji predstavljaju skriptu u *Python* programskom jeziku. Ta skripta predstavlja jedan od testova za ADAS. Koristi se i tvornica blokova u kojoj korisnik može izraditi vlastite blokove.

3.1.2 Node.js

Node.js je okruženje poslužitelja otvorenog koda koje koristi *JavaScript*. On je besplatan i može se koristiti na više platformi kao što su *Windows*, *Linux*, *Unix*, *Mac OS X* i slični [5]. *Node.js* koristi asinkrono programiranje te je korišten za građenje skalabilnih mrežnih aplikacija [6]. *Node.js* koristi asinkroni događajima pokrenut *JavaScript runtime* model. Takav model predstavlja

petlju događaja (engl. *event-loop*) kao *runtime* umjesto biblioteke. Uobičajeno ponašanje poslužitelja je definirano kroz povratne pozive (engl. *callbacks*) na početku skripte i na kraju pokreće poslužitelj putem blokirajućeg poziva. U *Node.js* nema takvog poziva za pokretanje petlje događaja. On jednostavno ulazi u petlju događaja nakon izvršavanja ulazne skripte. Zatim izlazi iz petlje događaja kada nema više povratnih poziva za izvršiti. Ovakvo ponašanje je slično ponašanju *JavaScript*-a u pregledniku, petlja događaja je skrivena od korisnika. On se koristi kako bi se stvorio dinamički sadržaj na web stranici, on može čitati, otvarati, stvarati, brisati, pisati i zatvarati datoteke na poslužitelju. *Node.js* može prikupljati podatke s poslužitelja. S njime se može dodavati, brisati i uređivati podatci u bazama podataka. *Node.js* datoteke sadržavaju zadatke koji će biti izvršeni nakon određenih događaja. Uobičajen događaj je kada netko pokušava pristupiti *port*-u na poslužitelju. *Node.js* datoteke moraju biti inicijalizirane na poslužitelju prije nego što se mogu koristiti ili prije nego što mogu imati nekakav utjecaj. One moraju imati nastavak „*.js*”. *Node.js* koristi takozvane module koji predstavljaju niz dodatnih funkcija koje nisu ugrađene u *Node.js*. Za instalaciju modula u *Node.js* koristi se tzv. *NPM* što je upravitelj paketima (engl. *package manager*) za *Node.js* ili upravitelj modulima. Paketi u *Node.js* sadrže sve datoteke potrebne za modul, a moduli su *JavaScript* biblioteke koje se mogu uključiti u vlastiti projekt. Preuzimanje i instalacija paketa se izvodi naredbom „npm install naziv_modula” [7]. Znanstveni rad u kojem se opisuje rad *Node.js* s *MongoDB* bazom podataka se proučava u navedenom radu [8].

3.1.3 MongoDB

MongoDB je distribuirana baza podataka orijentirana na dokumente. *MongoDB* pohranjuje podatke u fleksibilne dokumente slične *JSON*-u, što znači da se polja mogu razlikovati od dokumenta do dokumenta, a struktura podataka se može promijeniti tijekom vremena. Model dokumenata se može zapisati u aplikaciju kako bi jednostavnije upravljali dokumentima iz aplikacije. *Ad hoc* upiti, indeksiranje i skupljanje u stvarnom vremenu pružaju načine za pristup i analizu naših podataka. *MongoDB* je distribuirana baza podataka u svojoj srži, tako da su visoka dostupnost, horizontalno skaliranje i geografska distribucija su ugrađeni u bazu podataka i jednostavni su za korištenje [9]. *MongoDB* se može koristiti s *Node.js* i može pristupati bazi podataka i kolekcijama unutar baze podataka direktno iz web aplikacije.

3.1.4 Pug

Pug, prethodno poznat kao *Jade*, je predložak (engl. *template engine*) za *Node.js*, što znači da se podatci šalju s poslužitelja posredniku, *Pug*-u, te ih *Pug* prevede u *HTML* [10]. Sintaksa *Pug*-a je slična kao *HTML.* Razlikuje se od *HTML*-a u tome što nema oznaka za zatvaranje kod *Pug-a*. *Pug* koristi uvlačenje za određivanje gniježđenje oznaka. Postoje čak i skraćenice za klase (.) i ID (#). *Pug* može uključivati druge *Pug* datoteke kao *layout*, kako bi mogli dijeliti iste dijelove koda koji predstavljaju izgled vlastite stranice.

3.1.5 JavaScript

JavaScript (JS) je lagan, interpretiran ili preveden na vrijeme programski jezik. Lagani programski jezici su oni koji troše jako malo memorije, lagani su za implementirati i sadržavaju minimalističku sintaksu i svojstva. Interpretirani programski jezici su jezici kojima se linije koda izvode jedna za drugom, bez međustanja prevođenja jezika. Iako je najpoznatiji kao skriptni jezik za web stranice, mnoga okruženja koja nisu preglednici također ga koriste, poput *Node.js*, *Apache CouchDB* i *Adobe Acrobat*. *JavaScript* je prototipski jezik s višestrukom paradigmom, dinamični jezik, koji podržava stilove objektnog, imperativnog i deklarativnog stila (na primjer funkcionalnog programiranja) [11]. Korišten je za implementaciju kompleksnih funkcija na web stranicama. Svaki puta kada se želi na web stranici učiniti nešto više od prikazivanja statičnog sadržaja, koristi se *JavaScript*, kao npr. interaktivne karte, povlačenje datoteka s web stranica, animirana 2D ili 3D grafika i slično [12].

3.1.6 HTML

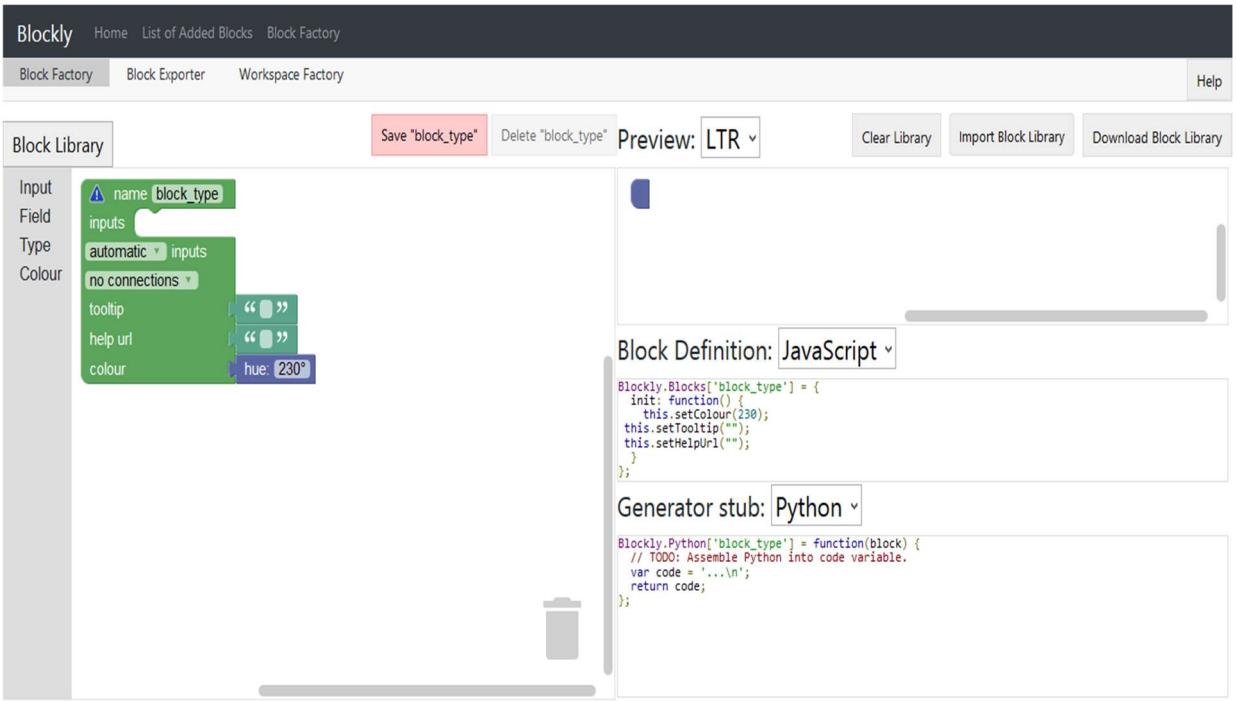
HTML je označni jezik (engl. *markup language*) koji se koristi za strukturiranje i davanje značenja vlastitom web sadržaju, npr. definiranje odlomaka, zaglavlja i tablica podataka, ili ugrađivanje slika i videozapisa na stranicu [13]. Označni jezici su jezici koji koriste oznake koje predstavljaju različiti sadržaj te se u njih mogu ubaciti dodatna svojstva. *HTML* se koristi za dizajniranje web stranica i se može kombinirati s drugim tehnologijama poput *CSS*, *JavaScript*, *AJAX*, *jQuery* i *PHP* kako bi naša web stranica sadržavala neke dodatne mogućnosti te kako bi ljepše izgledala krajnjem korisniku.

3.2 Opis novog rješenja za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju

Budući da je prethodno rješenje sadržavalo nekoliko nedostataka u samoj izvedbi koji su usporavali rad inženjera u stvaranju skripti za testiranje ADAS, tražilo se novo rješenje koje će ukloniti te nedostatke i ubrzati rad inženjera prilikom stvaranja skripti za testiranje ADAS .

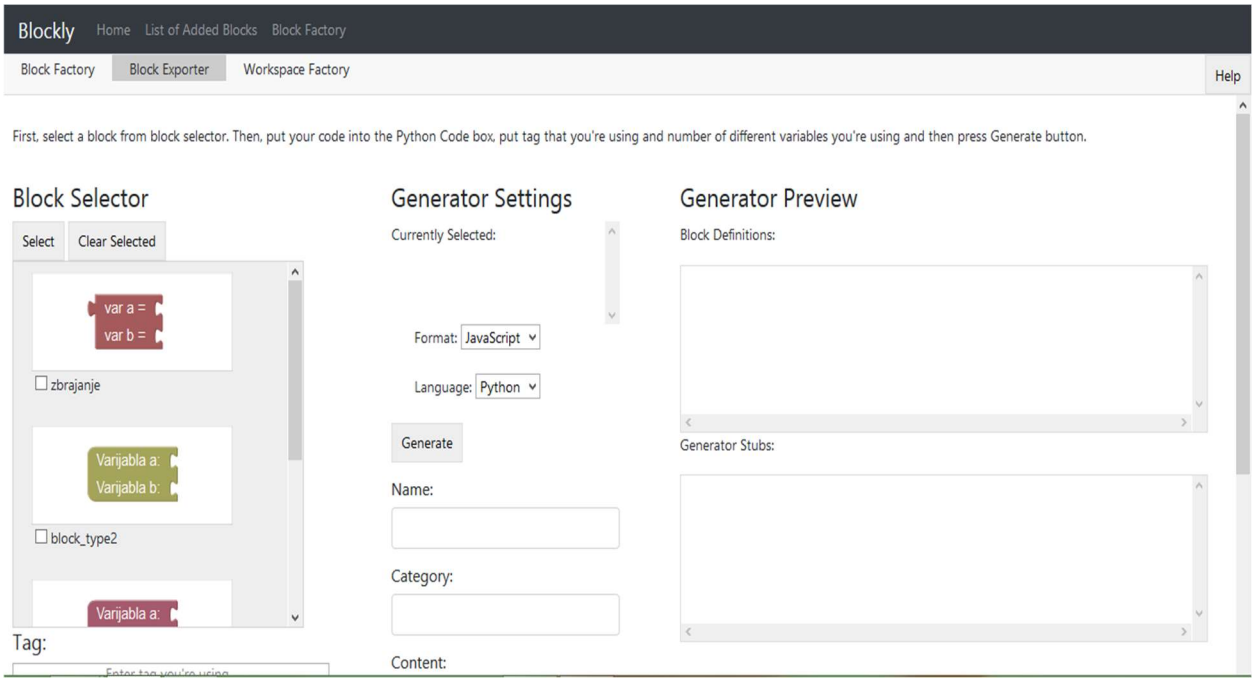
Novo rješenje za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju je zasnovano na ideji da se blokovi, koji se pišu i koji se koriste za generiranje skripti u programskom jeziku *Python,* moraju spremiti u bazu podataka. Budući da je bilo poznato da je *Blockly* napisan u *JavaScript*-u, tražilo se okruženje u kojem se također koristi *JavaScript* kako se ne bi miješali različiti programski jezici, također se tražilo da se to okruženje može povezati s bazom podataka. Zbog toga se koriste *Node.js* i *MongoDB*. *Node.js* i *MongoDB* su usko povezani s *JavaScript* jezikom te se njima može upravljati pisanjem naredbi u *JavaScript* jeziku. Novo rješenje je napisano u *Node.js* i spaja se na bazu podataka, točnije na kolekciju (engl. *collection*) dokumenata nazvanu *Blockly*, iz koje se učitavaju blokovi kao što bi *HTML* datoteka učitavala dodatne *JavaScript* ili *CSS* datoteke. Time se zadovoljio zahtjev spremanja i učitavanja blokova iz baze podataka (zahtjev 2 s popisa na početku 3. poglavlja). Idući korak bio je omogućiti stvaranje blokova koji će se kasnije koristi za stvaranje skripte za testove. Za *Blockly* se razvio alat zvan tvornica blokova pomoću kojeg korisnik može napraviti vlastite blokove za *Blockly* okruženje. Njegova integracija je zadovoljila zahtjev 4. Na slici 3.1 je prikazano kako alat tvornica blokova izgleda u novom rješenju. Do sada je izrađena mogućnost stvaranja i spremanja blokova na vlastitom poslužitelju no nije postojao neki brz i jednostavan način za pretvaranje *Python* koda u dio koda koji predstavlja funkcionalnost novog bloka. Taj problem je riješen korištenjem *Export* gumba u dijelu tvornice blokova koja je predviđena za preuzimanje programskog koda od jednog ili više izabranih blokova. Taj dio se naziva izvoznik blokova (engl. *Block Exporter*) i u njega je ubačen obrazac (engl. *form*) za izvlačenje potrebnog *Python* koda. Zatim je ubačen obrazac za unos bloka u bazu podataka. *Export* gumb je izmijenjen tako da raščlani (engl. *parse*) uneseni *Python* kod, da odvoji nazive varijabli koji sadrže nekakvu oznaku (engl. *tag*) i da sve to poveže s *Blockly* kodom od označenog bloka tako da se u obrascu za spremanje u bazu podataka dobije gotov kod vlastitog bloka koji je potpuno spreman da se spremi u bazu podataka. Ove funkcionalnosti zadovoljavaju zahtjeve 5 i 6 koji su navedeni na početku poglavlja. Spremljeni blok će sadržavati sve funkcionalnosti koje je korisnik htio, kao što je prikazano na slici 3.2.

Kada korisnik koristi novo-stvoreno rješenje, prvo što susretne je *Blockly* radni prostor u kojemu se nalazi gumb za prikaz *Python* koda kojeg blokovi u radnom prostoru predstavljaju (gumb „*Show Python*"), gumb za preuzimanje *Python* koda u obliku skripte na računalo (gumb „*Save Python*") i gumb za preuzimanje *Python* koda u obliku tekst datoteke na računalo (gumb „*Save Text*"), kao što je prikazano na slici 3.3. Kao dio radnog prostora nalazi se alatna kutija u kojoj se nalaze različite kategorije blokova koji se mogu koristiti za generiranje skripte za testiranje ADAS. Web stranica također sadrži poveznice (engl. *links*) koje mogu preusmjeriti korisnika na popis blokova koji se trenutno nalaze u bazi podataka („*List of Added Blocks*") i poveznicu na tvornicu blokova u kojoj možemo napraviti nove blokove („*Block Factory*").

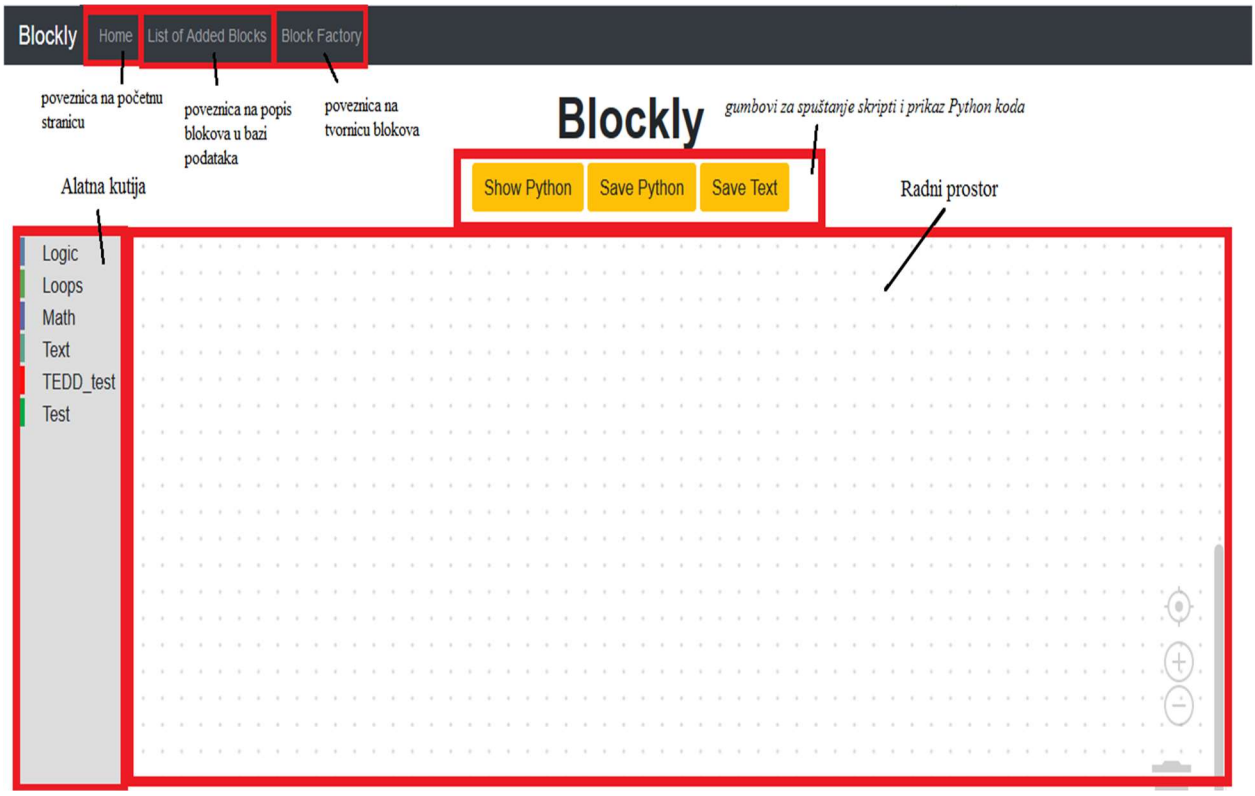


Slika 3.1. Prikaz grafičkog sučelja tvornice blokova za generiranje blokova u *Blockly* okruženju

Za pokretanje rješenja potrebno je imati instaliran *Node.js*, *MongoDB* s napravljenom kolekcijom na koju će se poslužitelj povezati. Dobro je imati naredbenu konzolu (engl. *console*) kao npr. *Git Bash* koji je bio korišten tijekom izrade novog rješenja za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju. Potrebno je imati sve mape i datoteke koje se koriste u programu kako bi mogli pokrenuti i koristiti rješenje.

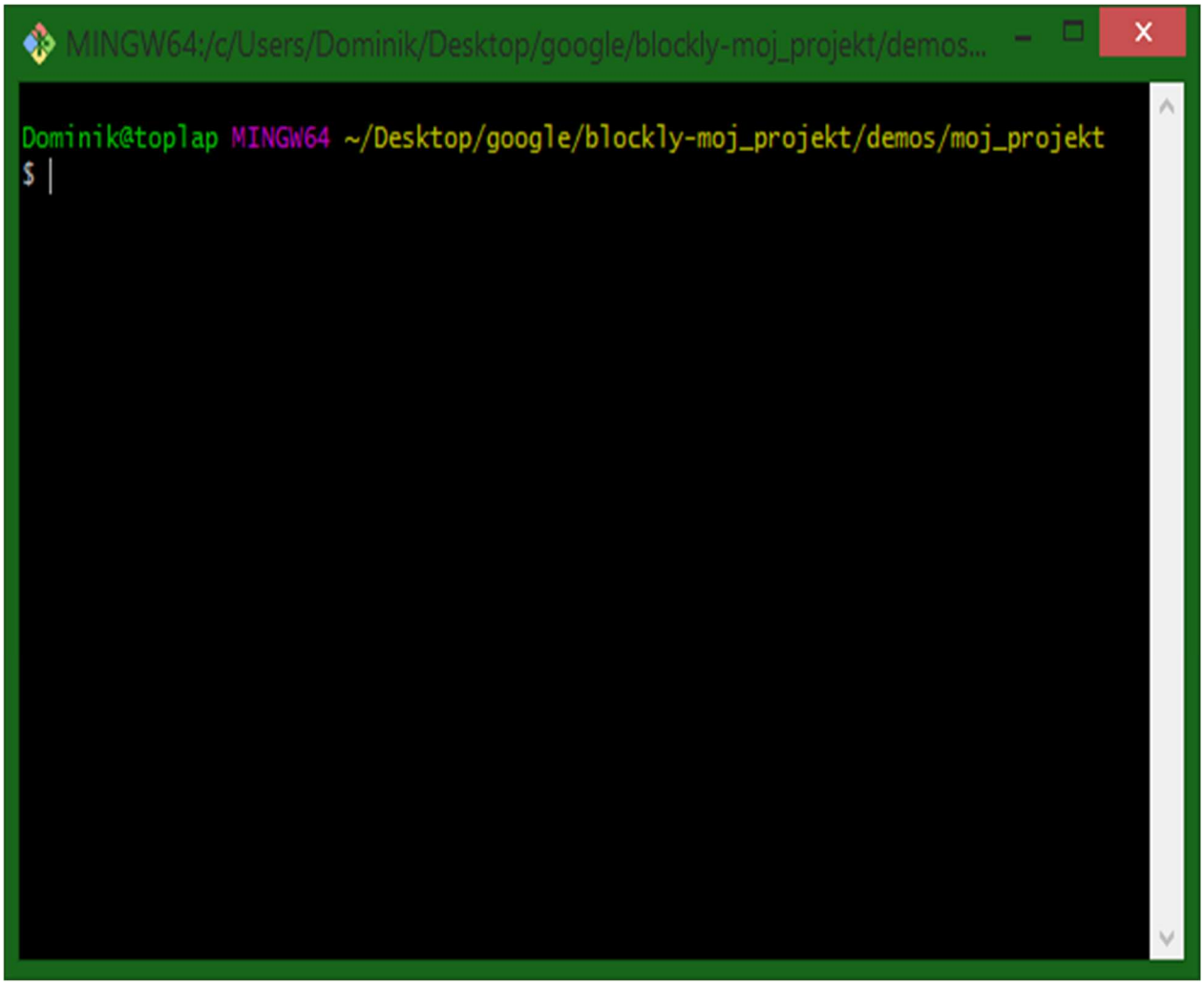


Slika 3.2. Obrazac za unos *Python* koda i obrazac za unos bloka u bazu podataka u *Blockly* okruženju



Slika 3.3. Prikaz grafičkog sučelja vlastitog rješenja za generiranje skripti za testiranje ADAS sustava zasnovanog na Blockly okruženju

Prije pokretanja programa potrebno se u konzoli pozicionirati u mapu gdje se nalazi *app.js* datoteka koja pokreće poslužitelj i upravlja radom poslužitelja, kao što je prikazano na slici 3.4.



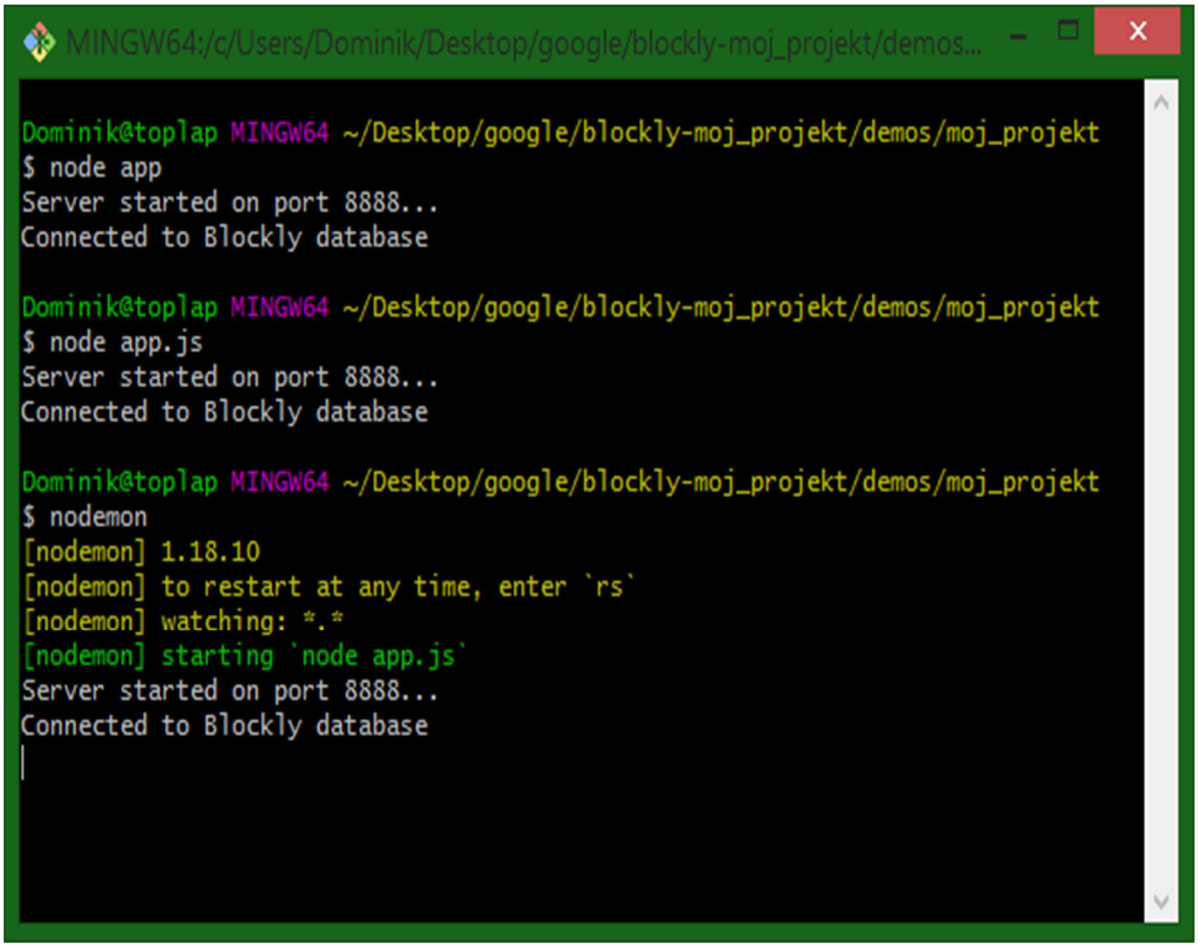
Slika 3.4. Prikaz naredbene konzole u kojoj je korisnik pozicioniran u mapu u kojoj se nalazi *app.js* datoteka

Nakon što je naredbena konzola otvorena i pozicionirana u mapi gdje se nalazi *app.js* datoteka, mogu se koristiti naredbe „*node app*", „*node app.js*" ili „*nodemon*", ali za korištenje naredbe „*nodemon"* treba biti instaliran *nodemon* i u datoteci *package.json* se treba navesti kako glasi naredba za pokretanje aplikacije, kao što je prikazano na slici 3.5.

Slika 3.5. Sadržaj *package.json* datoteke

Na slici 3.5 se može vidjeti kako se tu, osim naziva projekta i autora, nalazi puno više podataka. *Node.js* traži da se navedu osnovni podaci o projektu kada se pravi novi projekt. Također se mogu napisati neke zavisnosti (engl. *dependencies*) koje predstavljaju dodatne module koji se mogu instalirati pomoću *npm* ili upravitelja paketima. Na slici 3.6. se mogu vidjeti razlike prilikom pokretanja poslužitelja koristeći različite naredbe koje su prethodno navedene. Najduže vrijeme izvođenja, tj. podizanja poslužitelja ima „*nodemon*" naredba.

Na slici 3.6 je vidljivo da „*nodemon*” naredba pruža mogućnost ponovnog pokretanja poslužitelja i u slučaju greške ili promjene u kodu će pokušati ponovno pokrenuti poslužitelj. Naredbe „*node app*” i „*node app.js*” će prikazati greške ali neće pokušati ponovno pokrenuti poslužitelj, a ako se promijeni kod dok je poslužitelj pokrenut te naredbe neće primijetiti načinjene promjene nego je potrebno ručno ugasiti i upaliti poslužitelj.



Slika 3.6. Pokretanje poslužitelja koristeći različite naredbe (*node app*, *node app.js*, *nodemon*)

Nakon pokretanja poslužitelja se otvara vlastiti web preglednik i u adresnu traku se unosi „*localhost:8888*” (8888 je broj *port*-a koje rješenje koristi) ili ako se korisnik povezuje na poslužitelj s drugog računala onda je potrebno unijeti *IP* adresu računala koje je pokrenulo poslužitelj i broj *port*-a na kojem poslužitelj sluša (engl. *listening on port*), npr. „192.168.28.110:8888”. Kada se otvori stranica kao što je prikazano na slici 3.3 onda je korisnik pristupio rješenju i može ga koristiti.

Zahtjev 1 za rješenje je bio omogućiti preuzimanje *Blockly* radnog prostora kao *Python* skriptu i kao tekstualnu datoteku. Na slici 3.7 je prikazan kod koji omogućuje novo-dodanu funkcionalnost.

```
function download(text, name, type) // funkcija za spremanje preko id "a"
{
    var a = document.getElementById("a");
    var file = new Blob([text], {type: type});
    a.href = URL.createObjectURL(file);
    a.download = name;
}

function saveCode() // funkcija za spremanje blokova u python skriptu
{
    Blockly.JavaScript.INFINITE_LOOP_TRAP = null;
    var code = Blockly.Python.workspaceToCode(demoWorkspace);
    download(code, 'test_workspace.py', 'text/plain');
};

function downloadText(text, name, type) // funkcija za spremanje preko id "b"
{
    var b = document.getElementById("b");
    var file = new Blob([text], {type: type});
    b.href = URL.createObjectURL(file);
    b.download = name;
}

function saveText() // funkcija za spremanje blokova u text file
{
    Blockly.JavaScript.INFINITE_LOOP_TRAP = null;
    var texted_code = Blockly.Python.workspaceToCode(demoWorkspace);
    downloadText(texted_code, 'test_tekst.txt', 'text/plain');
}
```

Slika 3.7. Kod kojim je implementirana dodatna funkcionalnost za preuzimanje *Python* skripte i tekstualne datoteke

Kao što je prikazano na slici 3.7, funkcionalnost preuzimanja datoteke sastoji se od dvije odvojene funkcije napisane u *JavaScript* programskom jeziku. Jedna funkcija dohvaća element stranice preko njegovog *ID*, tj. identifikacijske oznake, zatim stvara nepromjenjiv objekt koji predstavlja sirov podatak tekstualnog tipa te ga povezuje preko poveznice. Druga funkcija pretvara blokove iz radnog prostora u *Python* kod te poziva funkciju koja je prethodno napisana i predajemo

joj tri parametra: varijablu u kojoj pretvaramo blokove u *Python* kod, naziv varijable s nastavkom koji predstavlja tip datoteke (ako je .py onda će biti *Python* skripta, ako je .txt onda će biti tekstualna datoteka) i treći parametar je *MIME* (engl. *Multipurpose Internet Mail Extensions*) tip koji predstavlja kako će preglednik procesirati *URL*. S ovom funkcionalnošću se zadovoljio zahtjev 1.

Zahtjev 2 bio je omogućiti povezivanje s bazom podataka, što se ostvarilo pravljenjem poslužitelja u *Node.js* okruženju. Za taj zahtjev je iskorišten modul *Mongoose* koji je napravljen za modeliranje objekata za *MongoDB* bazu podataka u *Node.js* te je napravljen kako bi pojednostavio upravljanje s objektima u *MongoDB* bazi podataka. Na slici 3.8. se vidi kod korišten za spajanje na bazu podataka pomoću naredbe „*connect*". Ona prima tri parametra: adresu u kojoj se nalazi baza podataka ili kolekcija na koju se želimo povezati, nekakve dodatne opcije koje želimo specificirati prilikom spajanja na bazu i povratnu funkciju koja se u ovom primjeru ne koristi. Nakon što je napravljena veza (engl. *connection*) s bazom podataka, napravljena je varijabla koja će koristiti tu vezu i pozivati funkcije za upravljanje s bazom. Ovdje se koriste funkcije „*once*" kako bi se znalo da je poslužitelj povezan s bazom i „*on*" kako bi poslužitelj primio i ispisao greške koje bi se mogle pojaviti. Ova funkcionalnost zadovoljava zahtjev 2.

```
mongoose.connect('mongodb://localhost/Blockly', { useNewUrlParser:true});
// Get Mongoose to use the global promise library
mongoose.Promise = global.Promise;
let db = mongoose.connection;

// Check Connection
db.once('open', function(){
    console.log('Connected to Blockly database');
});

// Check for DB errors
db.on('error', function(err){
    console.log(err);
});
```

Slika 3.8. Kod za spajanje na bazu podataka koristeći *Mongoose*

Do sada je napravljena mogućnost povezivanja s bazom podataka i napisane su osnovne funkcionalnosti za upravljanje elementima baze, tj. napisane su funkcionalnosti za upis, uređivanje

i brisanje elemenata iz baze podataka. Čitanje sadržaja, tj. blokova koji su upisani u bazu se napravilo na drugačiji način. Na slici 3.9 je vidljivo da se koristi „*/load_blocks*" putanja za učitavanje blokova iz baze podataka. To se radi zbog toga što je bilo potrebno iskoristiti praznu putanju na poslužitelju u koju bi se učitali blokovi i zatim ju povezati s web stranicom pomoću „<*script*>" oznake. Dalje vidimo kako se koristi model „*Block*" koji je tzv. *Schema* koju *MongoDB* i *Mongoose* koriste kako bi napravili predložak dokumenta koji se unosi ili koji se nalazi u bazi podataka. Preko „*Block*" modela se pristupa „*content*" atributu u kojem se skladišti sav kod potreban za jedan blok i učitava se taj atribut svakog elementa baze podataka na putanju „*/load_blocks*". Ova funkcionalnost je napravljena kao proširenje prethodne funkcionalnosti.

```
// Loading Custom Blocks
router.get('/load_blocks', function(req, res){
    // query which finds all blocks inside collection
    Block.find({}).select('content -_id').exec(function(err, blocks){
      if(err){
        console.log(err);
      }
      else {
        // Sends read database field to the client
        res.send(blocks.map(block => block.content).join('\n'));
      }
    });
});
```

Slika 3.9. Kod za učitavanje proizvoljnih blokova iz baze podataka

S obzirom na to da su za unos blokova i učitavanje blokova potrebni *JavaScript* datoteka i zapis svih blokova u *HTML* datoteku kao dio alatne kutije, bilo je potrebno napraviti način da se isto može odrađivati na poslužitelju bez intervencije korisnika. Kada korisnik unese blok u bazu podataka istovremeno se moraju upisati podaci o bloku u *HTML* datoteku. Na slici 3.10 je prikazano kako se riješio taj problem. Napravila se nova varijabla koja je objekt modela „*Block*". Njen naziv i sadržaj se dohvaća iz obrasca koji se nalazi na web stranici. Zatim se pristupa vlastitoj *HTML* datoteci pomoću modula koji upravlja datotečnim sustavom, *File System* modulom. S modulom *cheerio* se učitava cijela *HTML* datoteka i traži se alatna kutija na koju će se nadovezati podatci potrebni da *Blockly* prepozna blok. Ti podaci su naziv bloka i kategorija alatne kutije u

koju se želi svrstati vlastiti blok. Kategorija se unosi preko obrasca na web stranici i ona se dohvaća u varijablu. Nakon korištenja *cheerio* modula koristi se *Mongoose* kako bi se spremio novi blok u bazu podataka. Ovom funkcionalnošću se zadovoljio zahtjev 3. S ovime se u potpunosti riješio problem unosa blokova u bazu podataka, učitavanje blokova iz baze podataka i dinamični zapis u *HTML* datoteku kako bi se blokovi prepoznali na web stranici.

```
let block = new Block(); // stvara objekt koji je model Block
block.name = req.body.name; // sprema name iz onoga sto je uneseno na stranici
block.content = req.body.content; // sprema content iz onoga sto je uneseno na stranici
var categ = req.body.category;

fs.readFile('./index.html', 'utf8', function(err, data) {

  if (err){
    throw err;
  }

  // ucitavanje html-a i unos novog bloka u html
  var $ = cheerio.load(data);
  $('#'+categ).append('<block type="'+block.name+'" class="'+block.name+'"></block>').append('\n');

  fs.writeFile('./index.html', $.html(), function(err){
    if(err){
      console.log(err);
    }
    console.log("New block was successfully added and index.html was successfully overwritten!");
  });
});

block.save(function(err){
  if(err)
  {
    console.log(err);
    return;
  }
```

Slika 3.10. Kod za dodavanje novog bloka u bazu podataka

Zahtjev 4 bio je integrirati tvornicu blokova na poslužitelj. Ovaj zahtjev je u početku predstavljao problem jer se u preuzetoj mapi od *Blockly*-ja ne nalazi tzv. *closure-library* koji sadrži neke dodatne datoteke koje koristi tvornica blokova. *Blockly* naknadno omogućuje preuzimanje te mape i nakon što se ona preuzela, integriranje na poslužitelj nije predstavljalo problem. Na slici 3.11 se vidi kako se koristi nekoliko sličnih linija koda koje sadrže putanje. Ovdje se koristi funkcija vlastitog poslužitelja da statički posluži više mapa ili datoteka na poslužitelju prilikom otvaranja putanje na poslužitelju, tj. kada se pristupa putanji „*/block_factory*”. Tada se učitaju predane datoteke ili mape i prikaže se *HTML* datoteka koja sadrži tvornicu blokova i tada je tvornica blokova potpuno funkcionalna na poslužitelju. Zahtjev 4 je potpuno zadovoljen s ovom funkcionalnošću.

```
// Get Block Factory or Blockly Developer Tools
app.get('/block_factory', function(req, res){
  // serving static files for the Block Factory file
  app.use(serveStatic(path.join(__dirname + './closure-library/closure/goog/base.js')));
  app.use(serveStatic(path.join(__dirname, './blockfactory/')));
  app.use(serveStatic(path.join(__dirname, './blockfactory/workspacefactory/')));
  app.use(serveStatic('C:/Users/Dominik/Desktop/google/blockly-moj_projekt/appengine/storage.js'));
  app.use(serveStatic('C:/Users/Dominik/Desktop/google/blockly-moj_projekt/msg/js/'));

  // sending block-factory_index.html to the client
  res.sendFile('C:/Users/Dominik/Desktop/google/blockly-moj_projekt/demos/blockfactory/block-factory_index.html');
});
```

Slika 3.11. Kod za dodavanje *Block Factory* na poslužitelj

Jedini još preostali zahtjevi s popisa navedenog na početku ovog poglavlja bili su napraviti raščlanjivač (engl. *parser*) koda koji će iz unesenog *Python* koda izvući varijable koje sadrže oznaku i njih će zamijeniti s *Blockly* varijablama koje se nalaze u vlastitom bloku (zahtjevi 5 i 6) i napraviti gumb koji će povezati raščlanjeni kod s funkcionalnim kodom bloka prije upisivanja bloka u bazu podataka. Budući da su ti zahtjevi međusobno povezani, raščlanjivanje koda i povezivanje raščlanjenog koda s funkcionalnim kodom bloka je napravljen kao funkcionalnost jednog gumba. Slika 3.12 pokazuje kod korišten za povezivanje funkcionalnog dijela koda s raščlanjenim kodom. Varijabla *blockDef* predstavlja definiciju bloka, tj. dio koda kojeg *Blockly* stvori i koji predstavlja izgled bloka. Varijabla *firstHalf* predstavlja prvu polovicu koja je funkcionalni dio koda bloka koji se spaja s „*var code* =" jer u „*var code*" se unosi vlastiti proizvoljni kod koji je raščlanjen. Taj dio završava sa znakom za kraj reda i dodajemo varijablu *secondHalf* koja predstavlja završetak funkcionalnog dijela koda. Taj kod je podijeljen na dvije polovice kako bi jednostavnije i preciznije pristupili dijelovima koda koje je potrebno izmijeniti. Opisana funkcionalnost zadovoljava zahtjev 5 koji je naveden na početku poglavlja. Posljednja linija koda na slici 3.12 predstavlja element tekstualne kutije (engl. *textbox* ili *textarea*) koji prima vrijednost vlastitog spojenog koda dok se tekstualna kutija nalazi u obrascu za unos bloka u bazu podataka.

```
/* Putting together the whole string */
var genStubString = blocDef + "\n\n" + firstHalf + "var code = '" + pyCode3 + "';\n" + secondHalf;

//document.getElementById("py_code_box").value = genStubString;
document.getElementById("blockArea").value = genStubString;
```

Slika 3.12. Povezivanje funkcionalnog koda s raščlanjenim kodom

Slika 3.13 prikazuje potpunu logiku raščlanjivanja unesenog *Python* koda. Prolazi se kroz petlju koja je jednako duga koliko je dug korisnikov uneseni kod. Prvo se traži predana oznaka, zatim se ona uspoređuje s ostatkom koda. Kada se oznaka pronađe traži se gdje se varijabla nalazi u kodu i sastoji li se od jednog ili više znaka. Kada se varijabla pronađe traži se razmak (engl. *whitespace*), kraj niza znakova (engl. *string*) ili kraj koda nakon čega se varijabla izvuče i spremi kao novi element polja koji se zove *varNames*. Nakon što petlja prođe cijeli kod i izvuče sve varijable, provjerava ako su neke varijable više puta izvučene te se u novi objekt spreme jedinstveni nazivi varijabli kojima se kasnije doda „*value_*", što predstavlja prefiks koji varijable u *Blockly* blokovima sadrže. S ovom funkcionalnošću se zadovoljava zahtjev 6 koji je naveden na početku poglavlja. Ova funkcionalnost kao i funkcionalnost prikazana na slici 3.12 se pozivaju pomoću gumba *Generate* koji je prethodno bio nazvan *Export*. Kod prikazan na slikama 3.12 i 3.13 napisan je u datoteci *block_exporter_controller.js* koja se koristi za upravljanje funkcionalnostima tvornice blokova.

```
/* Parsing logic */
for(i = 0; i < pyCode4.length; i++) // it's looping through whole python code
{
  j = 0;
  while(j < userTag.length && (pyCode4.charAt(i+j) == userTag.charAt(j))) // it's checking whether the tag is found one character at a time
  {
    j++;
  }
  if(j == userTag.length) // if tag has been found
  {
    if((pyCode4.length - i - userTag.length) <= 10)
    {
      varNames.push(pyCode4.slice(i + userTag.length, pyCode4.length));
    }
    if((i + userTag.length) == 1) // if the variable we want is at the end of a string
    {
      varNames.push(pyCode4.slice(1, pyCode4.length));
    }
    else
    {
      // if variable is one letter store it into an array isprobao '<br/>' || , /([^>\r\n]?)(\r\n|\n\r|\r|\n)/
      if(pyCode4.charAt(i + userTag.length + 1) == (' ' || '' || '.'))
      {
        varNames.push(pyCode4.slice(i + userTag.length, i + userTag.length + 1));
      }
      // stores variable in an array if it's longer than one letter
      else
      {
        varNames.push(pyCode4.slice(i + userTag.length, pyCode4.indexOf(' ', i + userTag.length)));
      }
      // if tag is found and variable is sliced then swap whitespaces with full stops
      pyCode2 = pyCode.replace(" ", "."); // swaps whitespaces with full stops
      if(pyCode2.charAt(i) == ' ') // if whitespace is found then replace string with full stops
      {
        pyCode = pyCode2.substr(0, i) + '.' + pyCode2.substr(i, 1);
      }
      else
      {
        pyCode = pyCode2.replace(" ", ".");
      }
    }
  }
}
```

Slika 3.13. Logika raščlanjivanja (engl. *parsing logic*) koda

Rad na poslužitelju i stranice kojima se može pristupiti povezuju se preko putanja (engl. *routes*). Putanje su način kako poslužitelj može upravljati radom stranica i može javljati moguće greške korisniku ili programeru te omogućuju lagan pristup funkcionalnostima *Node.js* funkcija i modula koji su se naknadno instalirali. Bez putanja, GET i POST metoda poslužitelj ne može raditi. Budući da se koristi poslužitelj, vrlo je lagano povezati sve korištene sadržaje. Sadržaji poput HTML stranica i raščlanjivača koda se mogu koristiti zasebno, ali tada funkcije koje izvodi poslužitelj neće raditi. Funkcionalnosti koje ne bi mogle raditi bez poslužitelja su čitanje iz baze podataka i pisanje u bazu podataka, te izmjenjivanje podataka u bazi podataka. Poveznice na bilo koju od stranica ili putanja koje poslužitelj koristi neće biti funkcionalne bez rada poslužitelja. Ako bi se zasebno otvorila *HTML* datoteka koja se koristi za početnu stranicu ili *HTML* datoteka koja se koristi za generiranje blokova, raščlanjivanje i stvaranje koda te spremanje u bazu podataka, onda bi se mogle koristiti one funkcionalnosti koje se ne nalaze na poslužitelju.

4. VERIFIKACIJA ISPRAVNOSTI RADA NOVIH DODATNIH FUNKCIONALNOSTI

Ovo poglavlje opisuje način provjeravanja zadovoljenja zahtjeva i zadataka koji su se trebali riješiti dodanim funkcionalnostima izrađenim u sklopu ovog diplomskog rada.

4.1 Opis testova za provjeru rada implementiranih funkcionalnosti

U prethodnom poglavlju bili su navedeni zahtjevi koje je bilo potrebno zadovoljiti u novom rješenju i opisano je kako su oni zadovoljeni implementiranim funkcionalnostima u novom rješenju. Te funkcionalnosti su se testirale kako bi se provjerila ispravnost njihova rada. Budući da su neki zahtjevi bili međusobno povezani, onda su se implementirane funkcionalnosti koje su trebale zadovoljiti te zahtjeve zajedno i testirale.

Prvi testovi su se odnosili na to hoće li gumbovi „*Save Python"* za preuzimanje *Python* koda i „*Save Text*" za preuzimanje tekstualne datoteke uspješno raditi. Ovi testovi su testirali funkcionalnost koja je napravljena kako bi zadovoljila zahtjev 1. To se testiralo tako što su se kombinirali vlastiti blokovi s početnim blokovima koji se dobiju od *Blockly* okruženja. Ako su datoteke koje su preuzete bile prazne ili ako stranica nije pokrenula preuzimanje datoteke onda je rezultat testa bio neuspješan i morao se popraviti kod s kojim se preuzimaju datoteke. No ako su se datoteke preuzele kao *Python* skripta ili kao tekstualna datoteka s očekivanim kodom, onda je rezultat testa bio uspješan.

Zahtjev 2 je bio spajanje i učitavanje blokova iz baze podataka. Funkcionalnost zahtjeva 2 se testirala tako što se morala otvarati alatna kutija i ako se ne otvori ili ako se dobije nekakav rezultat koji nije bio očekivan, onda je rezultat testa bio neuspješan te se korištenjem konzole pretraživača dobila povratna informacija o grešci u kodu ili općenitoj grešci s putanjama ili problemima s učitavanjem. Sličan test se provodio kada se testirao rad učitavanja blokova i njihov zapis u HTML datoteku. Testiranje te funkcionalnosti je provjeravalo rad funkcionalnosti koja zadovoljava zahtjev 3. Budući da je klasičan način učitavanja blokova čitanje iz alatne kutije koja se nalazi u *HTML* datoteci, morao se pronaći drugi način kako bi se mogli učitati blokovi. To se obavilo tako što se u „*<script>*" oznaku unijela putanja u kojoj se učitavaju blokovi na poslužitelju, dok se testiranje provodilo na isti način kao i prethodni test. Provjeravalo se jesu li podaci učitani tako što se odlazilo na tu lokaciju, tj. putanju u kojoj su se učitavali ti podatci. Kako se ne bi moralo ručno pisati u HTML datoteku kategorija i naziv bloka, razvio se način da to poslužitelj sam

napravi prilikom unosa bloka u bazu, zatim se točnost toga provjerila istom metodom. To se testiralo tako što se na početnoj stranici pokušala otvoriti alatna kutija i ako se ona otvorila uspješno onda se pokušala otvoriti kategorija u kojoj se nalazi novi blok. Ako se kategorija nije uspješno otvorila onda se otkrio problem, no ako se kategorija uspješno otvorila onda je rezultat testa bio uspješan.

Budući da rad s bazom podataka ne znači samo biti povezan s bazom podataka nego i pisanje, brisanje i izmjenjivanje podataka, onda su se te funkcionalnosti morale provjeriti. Ovi testovi su se odnosili na sam upis, brisanje i izmjenjivanje podataka u bazi podataka (zahtjev 2). U rješenju se nalazi stranica u kojoj se učitavaju i prikazuju svi blokovi koji se nalaze u kolekciji koja se koristi. Budući da je *MongoDB* baza podataka orijentirana na dokumente, onda su njezine tablice nazvane kolekcije jer ne može postojati tablica dokumenata, no postoji kolekcija dokumenata. Budući da na poslužitelju postoji popis svih blokova, koristeći program *MongoDB Compass* se provjeravalo podudaraju li se uneseni blokovi u bazi i u rješenju, te su se pokušale napraviti izmjene na samim blokovima i uspoređivalo se hoće li se blokovi i izmjene pojaviti u tom programu. *MongoDB Compass* je program koji se spaja na *MongoDB* bazu podataka i ima pristup svim kolekcijama dokumenata te on predstavlja grafičko sučelje s kojim možemo pristupati našim dokumentima, tj. sadržaju baze podataka. Ako su se podatci u vlastitom rješenju za generiranje skripti za testiranje ADAS zasnovanog na *Blockly* okruženju podudarali s podatcima u programu *MongoDB Compass* onda nam je rezultat testa bio uspješan, ako nisu onda je rezultat testa bio neuspješan.

Testiranje funkcionalnosti koja bi trebala zadovoljiti zahtjev 4 je provjeravalo koliko je dobro integrirana tvornica blokova na poslužitelj. To se testiralo tako što se otvarala putanja „*/block_factory*" u kojoj se nalazi tvornica blokova. Ako su postojali problemi prilikom učitavanje te putanje, u konzoli web preglednika su postojale greške i upozorenja, onda je rezultat testiranja bio neuspješan i te su se greške i upozorenja ispravila te se testiranje ponovilo. Nakon što su se sve greške i upozorenja ispravila, rezultat testiranja je bio uspješan.

Posljednja vrsta testova se odnosila na funkcionalnost raščlanjivanja i proizvodnju koda (engl. *parsing and generating code*) koja zadovoljava zahtjeve 5 i 6. Budući da se morao razviti način koji će omogućiti inženjeru da unese programski kod napisan u *Python* programskom jeziku te da se iz tog koda izvuku riječi s oznakama i ubace varijable koje koristimo u bloku umjesto njih, onda se razvio raščlanjivač (engl. *parser*) koji je to radio. Testovi su se provodili tako što se

napisalo 7 različitih skripti koje su po sadržaju i po funkciji bile različite. Skripte su bile napisane na drugačiji način kako bi se različite situacije mogle testirati. Bilo je potrebno koristiti različite testove kako bi se mogao napisati raščlanjivač koji radi u svakom slučaju. Ako raščlanjivač raščlani jednu varijablu no ne raščlani drugu onda nešto ne radi, ako raščlani obje no napravi to na krivi način onda opet ne radi i tako dalje. Proizvođač koda je integriran s raščlanjivačem tako da kada korisnik pritisne *Generate* gumb, istovremeno se raščlani kod i proizvede se novi, spojeni kod. Proizvođač koda je tijekom svakoga testa radio kako je zamišljeno te se njegov rad nije morao dodatno testirati.

Testiranje funkcionalnosti gumbova „*Show Python*", „*Save Python*" i „*Save Text*" nije bilo uspješno u početku testiranja. Rezultati testova nisu bili uspješni jer kod koji je upravljao funkcionalnosti nije bio dobro napisan. Nakon istraživanja na tu temu i nakon što se kod izmijenio, funkcionalnost gumbova „*Show Python*", „*Save Python*" i „*Save Text*" je bila zadovoljavajuća, čime su rezultati testova postali uspješni.

Nakon testiranja svih funkcionalnosti na načine opisane ranije, ustanovilo se da sve funkcionalnosti rade u potpunosti, tj. rezultati testova svih funkcionalnosti su bili uspješni.

4.2 Usporedba novog rješenja s dodanim funkcionalnostima s prethodnim rješenjem

Iz prethodnog rješenja nastali su zahtjevi čije bi zadovoljenje trebalo omogućiti jednostavnost korištenja programa te bi olakšalo rad korisniku pri stvaranju vlastitih blokova koji bi se kasnije koristili za stvaranje vlastitih *Python* skripti i olakšalo bi korisniku skladištenje tih blokova tako da se oni nalaze na jednom mjestu. Ti su zahtjevi zadovoljeni u novom rješenju tj. rješenju s dodanim novim funkcionalnostima. Povezivanje s bazom podataka te skladištenje blokova i pravljenje blokova s unesenim *Python* kodom su funkcionalnosti koje prethodno rješenje nema. Rad korisnika je znatno brži i učinkovitiji kada se koristi rješenje unaprijeđeno u sklopu ovog rada jer na web stranici vlastitog rješenja postoji mogućnost izrade bloka sa svojim kodom napisanim u *Python* jeziku te nakon što se spremi blok u bazu podataka, može se i koristiti. U prethodnom rješenju taj se posao izvodio tako što su se morali koristiti *Blockly* razvojni alati lokalno ili na internetu, dok su u novom rješenju oni integrirani. Nakon što se napravi blok u jednu datoteku se sprema kod za izgled novog bloka dok se u drugu datoteku sprema kod za funkcionalnost bloka u koji se tek treba unijeti vlastiti kod.

4.3 Potencijalne buduće dorade stvorenog rješenja

U skladu sa zahtjevima postavljenim na dodatne funkcionalnosti prije izrade diplomskog rada, postojeće rješenje je unaprijeđeno i zahtjevi su u potpunosti zadovoljeni. Prilikom izrade dodatnih funkcionalnosti došlo se do ideja što bi se sve potencijalno moglo još u budućnosti dodati u rješenje, a što bi ga dodatno unaprijedilo. Tako bi se primjerice mogla dodati nekakva vrsta tražilice u koju bi korisnik unosom naziva bloka dobio blokove koji sadrže taj naziv. Uz to, mogao bi se dodati i sustav registracije i prijave korisnika kako bi cijelo rješenje bilo sigurnije jer bi ograničilo pristup samo ovlaštenim korisnicima. Mogao bi se i ograničiti pristup na desetak do dvadesetak korisnika, kako bi smanjilo opterećenje poslužitelja te omogućilo brz i uzvratni (engl. *responsive*) rad.

5. ZAKLJUČAK

Testiranje programske podrške ADAS općenito znači pisanje velikog broja skripti koje će temeljito testirati sustav i dati povratnu informaciju. Budući da se pisanjem takvih skripti često ponavljaju iste funkcije i dijelovi koda, u sklopu ovog rada bilo je potrebno razviti okruženje koje će skratiti vrijeme stvaranja skripti. Kod toga je uvelike pomogao *Blockly*, no tu je nastao problem jer se treba napraviti veliki broj blokova koji predstavljaju programski kod. Stoga je bilo potrebno razviti rješenje koje će skladištiti blokove u bazu podataka i olakšati inženjeru unos bloka i način prevođenja unesenog koda u blok.

Cilj diplomskog rada je bio izraditi okruženje u obliku web poslužitelja koji će koristiti *Blockly*, biti povezano s bazom podataka, moći stvarati nove blokove i unijeti programski kod napisan u *Python* jeziku u blok te unijeti takve blokove u bazu podataka. Programsko rješenje je web poslužitelj na kojemu se koriste dvije *Blockly* stranice od kojih se jedna koristi za izradu *Python* skripti dok druga stvara blokove i upisuje ih u bazu podataka. Programsko rješenje je napravljeno tako da korisnik na jednom mjestu može napraviti blokove i upisati ih u bazu podataka. Programsko rješenje je razvijeno u *Node.js* tehnologiji i povezuje se s *MongoDB* bazom podataka. Zadatak je u potpunosti izvršen, što su provedeni testovi ispravnosti rada svih funkcionalnosti i pokazali. Konačno, programsko rješenje sadrži sve funkcionalnosti koje su bile navedene u zahtjevima prije izrade samog rada. Kreirano unaprijeđeno rješenje korisniku može poslužiti kao alat za generiranje velikog broja skripti koje će predstavljati testove za ADAS. Unaprijeđeno rješenje omogućuje znatno brži rad nego postojeće rješenje koje je opisano u drugom poglavlju. Ono omogućuje inženjeru da brzo i na jednom mjestu napravi sve blokove koji su mu potrebni te da ih kombinira u *Python* skripte pomoću kojih testira ADAS.

LITERATURA

[1] D. Kum, J. Son, S. Lee, I. Wilson, Automated Testing for Automotive Embedded Systems, SICE-ICASE International Joint Conference, str. 4414 – 4418, Busan, Koreja , 2006

[2] S. A. Al-Arian, M. Nordenso, FUNTEST: a functional automatic test pattern generator for combinational circuits, Proceedings. 'Meeting the Tests of Time'., International Test Conference, str. 945 – 946, Washington, DC, USA, 1989

[3] Global Edgesoft, http://www.globaledgesoft.com/blog-posts/what-is-adas/ , pristupio 23.7.2019

[4] Blockly, https://developers.google.com/blockly/guides/overview , pristupio: 1.7.2019

[5] W3Schools - Node.js, https://www.w3schools.com/nodejs/nodejs_intro.asp , pristupio: 1.7.2019

[6] Node.js – About, https://nodejs.org/en/about/ , pristupio: 1.7.2019

[7] W3Schools – Node.js NPM, https://www.w3schools.com/nodejs/nodejs_npm.asp , pristupio: 1.7.2019

[8] D. Zhang, S. Lin, Y. Fu, S. Huang, The communication system between web application host computers and embedded systems based on Node.JS, 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI 2017), Shanghai, Kina, 2017

[9] MongoDB, https://www.mongodb.com/what-is-mongodb , pristupio: 1.7.2019

[10] What is Pug.js (Jade) and How can we use it within a Node.js Web Application?, https://codeburst.io/what-is-pug-js-jade-and-how-can-we-use-it-within-a-node-js-web-application-69a092d388eb , pristupio: 1.7.2019

[11] JavaScript MDN, https://developer.mozilla.org/en-US/docs/Web/JavaScript , pristupio: 1.7.2019

[12] What is JavaScript?, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript , pristupio: 1.7.2019

[13] W3Schools – What is HTML?, https://www.w3schools.com/whatis/whatis_html.asp , pristupio: 1.7.2019

SAŽETAK

Na osnovu detektiranih nedostataka postojećeg rješenja za generiranje skripti za testiranje ADAS zasnovanog na Blockly okruženju, definirani su zahtjevi za funkcionalnostima koje novo rješenje za istu namjenu mora zadovoljavati. Zahtjevi su uključivali: zapis blokova u bazu podataka, mogućnost preuzimanja tekstualne datoteke i *Python* skripte, dinamičko zapisivanje kategorije i naziva bloka u alatnu kutiju u *HTML* datoteci, integracija tvornice blokova u aplikaciju, mogućnost unosa *Python* koda i raščlanjivanje varijabli s oznakom te zamjenjivanje tih varijabli s varijablama koje blok koristi i stvaranje gumba koji koristi raščlanjeni kod i povlači kod koji predstavlja izgled i funkcionalnost bloka te ubacuje raščlanjeni kod u funkcionalni dio bloka. Svi su zahtjevi riješeni, a provedeni testovi pokazali su uspješnost rada svih dodanih funkcionalnosti. Nove funkcionalnosti omogućuju korisniku znatno brži rad nego postojeće rješenje i omogućuje na jednom mjestu generiranje svih blokova potrebnih za stvaranje *Python* skripti pomoću kojih se testira ADAS.

Ključne riječi: testiranje, Node.js, Blockly, JavaScript, baza podataka

GRAPHIC ENVIRONMENT FOR GENERATING AUTOMATIC TESTS AND DOCUMENTATION FOR ADAS

ABSTRACT

Based on the detected deficiencies of the existing program solution for creating scripts for testing ADAS based on *Blockly* environment, requirements for functionalities which the new solution for the same purpose must meet are defined. Requirements included: writing block to a database, ability to download text files and *Python* scripts, dynamically writing category and block names in a toolbox in an *HTML* file, integrating a *Block Factory* into an application, ability to enter *Python* code and parse variables with a tag, and replace those variables with the variables the block uses and create a button which uses parsed code and pulls code which represents the block's appearance and functionality and inserts the parsed code into the functional part of the block. All requirements were resolved, and the tests performed showed the performance of all the added functionalities. The new functionalities enable the user to perform much faster than the existing program solution and allows to build all the blocks needed to create *Python* scripts that test ADAS at one place.

Keywords: testing, Node.js, Blockly, JavaScript, database

ŽIVOTOPIS

Dominik Birtić je rođen u Osijeku, Republika Hrvatska, 6. siječnja 1996. godine. Pohađao je osnovnu školu „Retfala“ u Osijeku i svih osam razreda prošao je s odličnim uspjehom. Nakon osnovne škole, 2010. godine upisuje prirodoslovnu-matematičku gimnaziju u Osijeku i završava sve razrede s vrlo dobrim uspjehom. 2014. godine je maturirao s dobrim uspjehom. 2014. godine upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo. Preddiplomski studij završava 2017. godine i upisuje diplomski studij na istom fakultetu.

_____

Dominik Birtić