Praćenje vremena trajanja testova u razvoju programske podrške u automobilskoj industriji

Ivanović, Denis

Master's thesis / Diplomski rad

2022

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:200:732140

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

Sveučilišni studij

PRAĆENJE VREMENA TRAJANJA TESTOVA U RAZVOJU PROGRAMSKE PODRŠKE U AUTOMOBILSKOJ INDUSTRIJI

Diplomski rad

Denis Ivanović

Osijek, 2022.

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

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 16.09.2022.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Denis Ivanović
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1123R, 13.10.2020.
OIB studenta:	21223954683
Mentor:	izv. prof. dr.sc. Josip Job
Sumentor:	,
Sumentor iz tvrtke:	Davor Kedačić
Predsjednik Povjerenstva:	Doc. dr. sc. Denis Vranješ
Član Povjerenstva 1:	izv. prof. dr.sc. Josip Job
Član Povjerenstva 2:	Izv.prof.dr.sc. Ratko Grbić
Naslov diplomskog rada:	Praćenje vremena trajanja testova u razvoju programske podrške u automobilskoj industriji
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Hardver koji se koristi u automobilskoj industriji često je specifičan i ograničeno dostupan pa je samim time i otežano njegovo testiranje u smislu da je moguće provesti velik broj različitih testova u što kraćem vremenu na zadovoljavajući način. Ispitivanje programske podrške za takvu vrstu hardvera moguće je, uz određene preduvjete, provoditi s udaljene lokacije stoga je mjerenje vremena utrošenog na pojedine testove od velike važnosti za razvoj programske pordrške u takvim situacijama. Mjerenje vremena utrošenog na testove pruža uvid u zauzeće resursa, a što je preduvjet za prikupljanje znanja koje u konačnici omogućava lakše i preciznije planiranje i provedbu radnih
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: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	16.09.2022.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2022.

Ime i prezime studenta:	Denis Ivanović
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1123R, 13.10.2020.
Turnitin podudaranje [%]:	4

Ovom izjavom izjavljujem da je rad pod nazivom**: Praćenje vremena trajanja testova u razvoju programske podrške u automobilskoj industriji**

izrađen pod vodstvom mentora izv. prof. dr.sc. Josip Job

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:

SADRŽAJ

1. UVOD

Prema istraživanjima koje svake godine radi Svjetska organizacija proizvođača motornih vozila (engl. *International Organization of Motor Vehicle Manufacturers*) jasno je vidljivo da je od 1950. godine (od kada se vode mjerenja) pa do danas, proizvodnja automobila u skoro pa kontinuiranom rastu, a samo u 2021. godini u svijetu je proizvedeno nešto više od 79 milijuna automobila[1]. Danas se u većini automobila nalazi od 30 do 50 mikrokontrolera [2], pa je samim time potreba za testiranjem programske podrške u automobilskoj industriji sve veća i zahtjevnija. Vrijeme koje se iskorištava za testiranje programske podrške u automobilskoj industriji je jako „skupo“. Iz tog razloga je vrlo značajan svaki napor da se testovi, testni slučajevi i vrijeme testiranja maksimalno smanje i optimiziraju. Osim velikih troškova testiranja, hardver u automobilskoj industriji je specifičan, ograničeno dostupan i skup, pa i to dodatno otežava testiranje u smislu brzine njegovog izvođenja, količine testova koje se mogu izvesti i točnosti tako izvedenih testova.

Prema podatcima iz studija Nacionalnog instituta za standarde i tehnologiju (engl. *National institute of standards and technology*) iz 2002. godine, neadekvatno upravljanje ili neispravno odabrani alati za upravljanje testiranjem mogu u velikoj mjeri otežati i poskupiti cijeli proces. Godišnje, ovi problemi gospodarstvu Sjedinjenih Američkih Država nanose štetu 22.2 – 59.5 milijardi dolara.[3]

Stoga je cilj svake tvrtke ili korporacije koja se bavi testiranjem, pažljivo mjeriti vrijeme i resurse koji su utrošeni za testiranje, da bi se onda ti podaci mogli kvalitetno obraditi i prikazati odgovornim osobama s ciljem bolje organizacije, preciznijeg planiranja i optimiziranijeg korištenja resursa prilikom izvođenja. Tema i zadatak ovog diplomskog rada je upravo ovaj segment problema, praćenje vremena trajanja testova i njihova predodžba na vizualno smislen način.

U drugom poglavlju ovog rada razmotreno je područje koje se obrađuje ovom temom, a to su alati za upravljanje testiranjem. Objašnjeni su neki od najkorištenijih alata, kao i razlog razvoja novog alata kroz obradu ovog diplomskog rada. Nadalje, u trećem poglavlju kratko su prikazani osnovni zahtjevi koje treba sadržavati pokazni softver kako bi što bolje pokazao na koji način radi primarni softver razvijen kroz rad u tvrtki *TTTech-Auto*. U četvrtom poglavlju je kroz tri potpoglavlja detaljno objašnjeno programsko rješenje pokaznog softvera, a svako potpoglavlje reprezentira jedan glavni modul sustava, dok svaki modul predstavlja jednu stanicu u internetskom pregledniku.

2. PREGLED PODRUČJA TEME

U drugom poglavlju ovog rada razmotreno je područje koje se obrađuje ovom temom. Alati za upravljanje procesom testiranja i praćenjem trajanja testova su programi koji omogućavaju da menadžeri i QA-inženjeri na najbolji mogući način koncipiraju potrebne testne slučajeve i što bolje organiziraju izvođenje testiranja. Danas ima puno komercijalnih alata za praćenje izvršenja testova i njihovog trajanja.

2.1. POSTOJEĆA PROGRAMSKA RJEŠENJA

Neki od najpoznatijih alata za upravljanje izvršavanjem testova su:

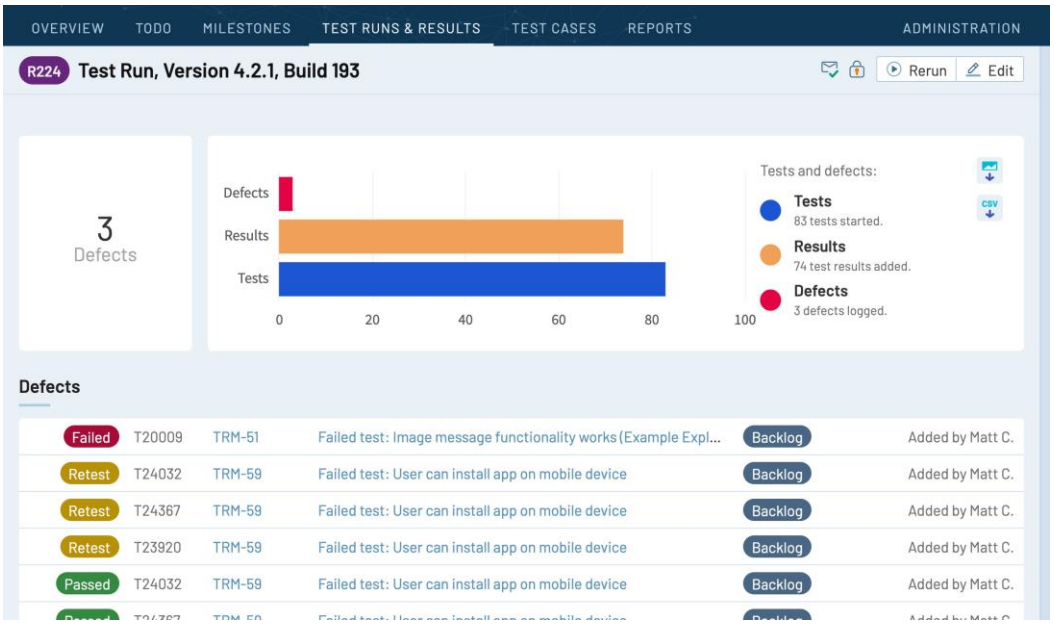
- *TestRail*
- *PractiTest*
- *Zephyr*
- *Kualitee*
- *TestCollab.*

Problem kod gotovih alata je taj što ih je gotovo nemoguće prilagoditi, ako je to potrebno. Zato sve više tvrtki i korporacija koje se primarno bave testiranjem ili one koje dosta testiraju i imaju potrebu za praćenjem vremena trajanja testova, učinkovitosti testera i uspješnosti testiranja, okreću razvoju internog alata.

TestRail – alat za upravljanje testnim slučajevima koji se upotrebljava putem internetskog sučelja. Podržava agilno i tradicionalno testiranje.[4] Na slici 2.1 prikazan je izgled sučelja *TestRail* softvera.[5]. Neke od glavnih značajki su:

- jednostavno praćenje statusa pojedinačnih testova
- lako mjerljiv napredak uz pomoć nadzornih ploča
- mogućnost generiranja izvještaja o aktivnostima
- praćenje opterećenosti tima i prilagođavanje zadataka i resursa
- opcije korištenja preko oblaka ili lokalno
- kompatibilan s raznim alatima za praćenje kvarova, primjerice *Atlassian Jira, FogBugz, Axosoft, GitHub* i *TFS*
- kompatibilan s vodećim alatima za automatizaciju testiranja.

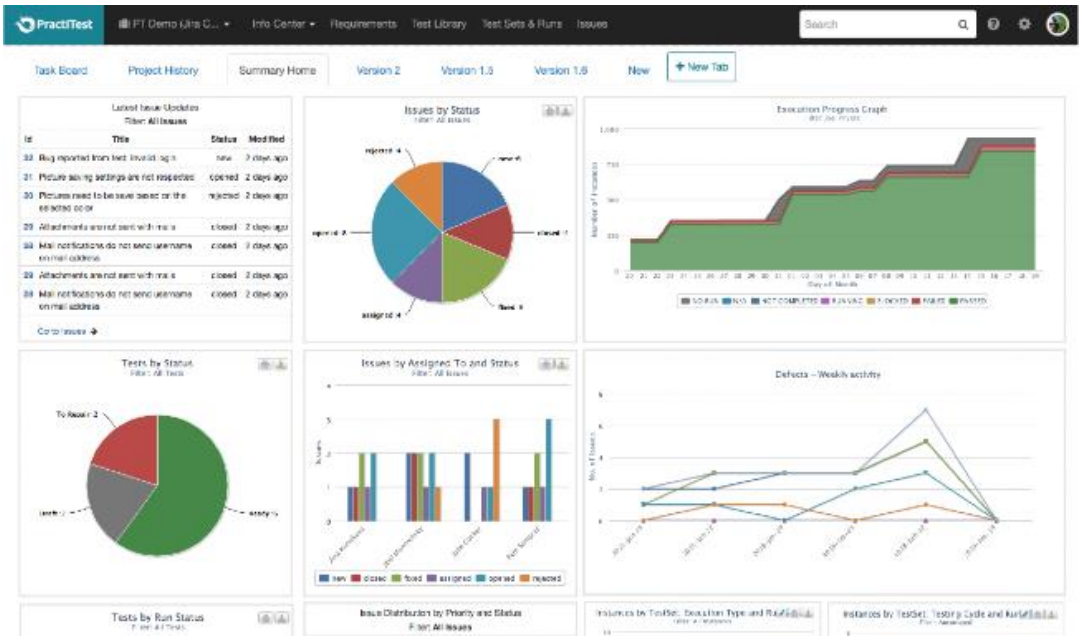
Jedna od rijetkih mana softvera *TestRail* je što on može biti teško povratiti izbrisane testne slučajeve. [6]



Sl. 2.1. Prikaz izgleda sučelja *TestRail* softvera.

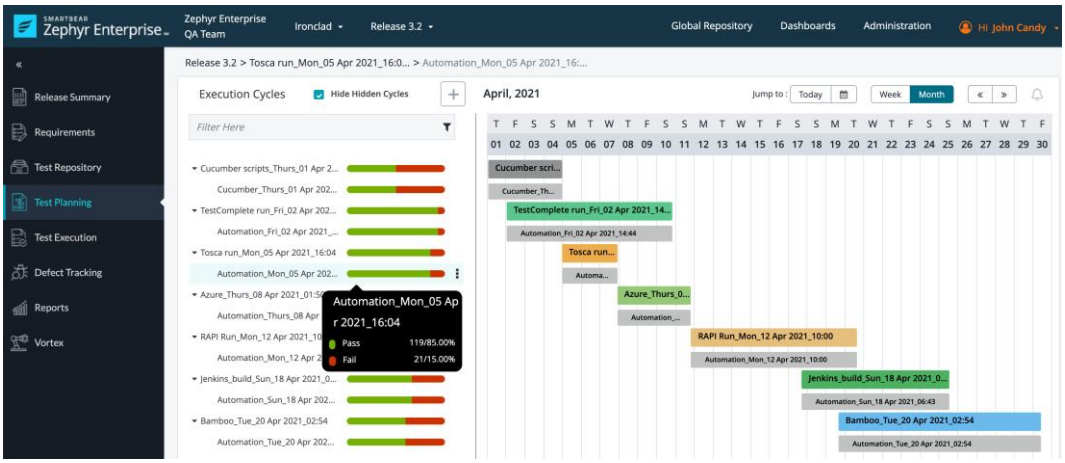
PractiTest – alat koji omogućuje upravljanje životnim ciklusom softvera. Omogućuje kreiranje i upravljanje zahtjevima na softver, kao i njihov uvoz iz npr. *Microsoft Excela*.[7] Na slici 2.2 prikazan je izgled sučelja softvera *PractiTest*.[8] Omogućuje automatsko testiranje, dobar uvid u napredak istog i uvid u riješenosti kvarova. Neke od glavnih značajki *PractiTesta* su:

- kompatibilnost i integrabilnost s uobičajenim alatima za praćenje grešaka i alatima za automatizaciju
- omogućuje prilagodbu polja, prikaza, ovlaštenja itd.
- koristi se rezultatima testova iz prethodnih kampanja i sesija
- dobar i pregledan hijerarhijski prikaz
- kontrolira unos defekata tako da se nikada ne unose i ne testiraju duplo
- pregledna upravljačka ploča s metrikama i kvalitetni izvještaji.



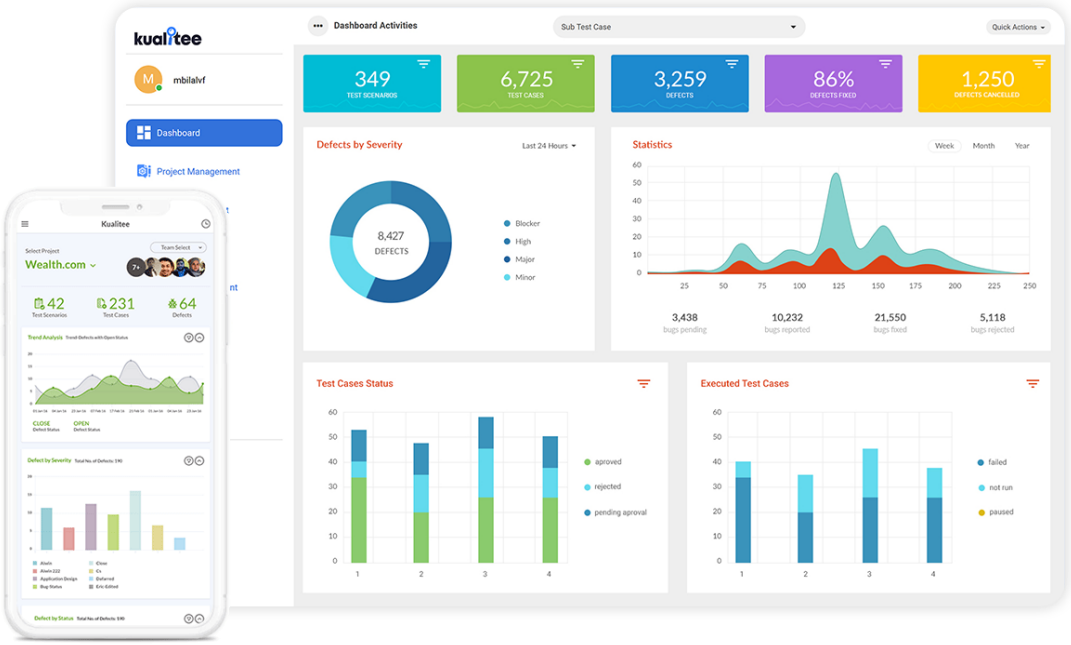
Sl. 2.2. Prikaz izgleda sučelja *PractiTest* softvera.

Zephyr – kao i ostali navedeni alati, i *Zephyr* odlikuju značajke dobrog upravljanja testovima, mogućnost automatiziranog testiranja i odlično upravljanje izvještajima. Na slici 2.3 prikazano je sučelje softvera *Zephyr*. [9] Jedna od razlika *Zephyra* u odnosu na prethodna rješenja je ta što *Zephyr* nema jednu bitnu značajku sigurnosti, a to je kontrola pristupa temeljena ulogama. Također, prema izvještajima korisnika, još jedna od većih zamjerki na ovaj sustav je ta što je spor za korištenje. [10]



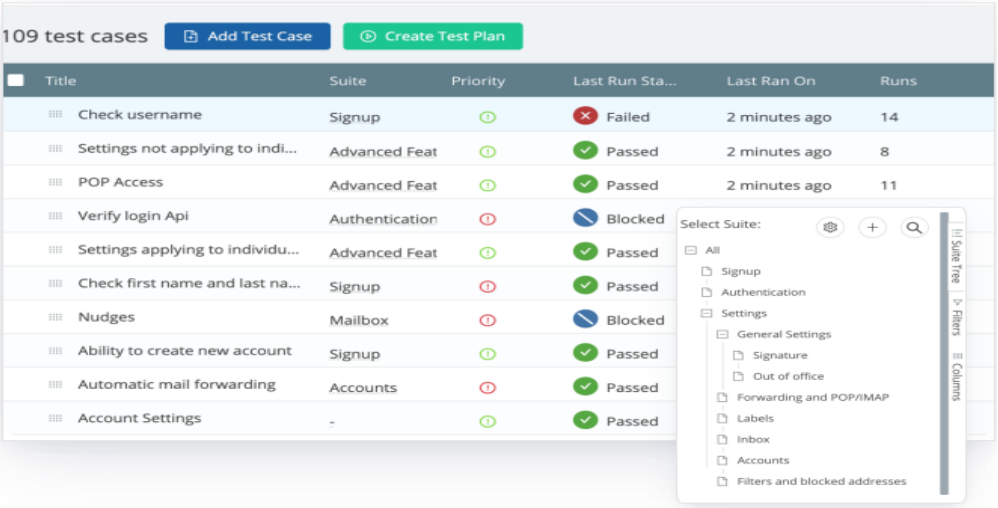
Sl. 2.3. Prikaz izgleda sučelja *Zephyr* softvera.

Kualitee – uz upravljanje testovima daje i mogućnost upravljanja projektima. Pored osnovnih funkcionalnosti koje ima većina alata, najveća mu je prednost upravljanja kompletnim projektima, nedostatcima i testovima pod jednim sučeljem. Na slici 2.4 prikazano je sučelje *Kualitee* softvera. Kao i *Zephyr* korisnici kao manu najviše navode to što je spor.[11]



Sl. 2.4. Prikaz izgleda sučelja *Zephyr* softvera.

TestCollab – posjeduje osnovne značajke kao i većina komercijalnih alata, ima mogućnost povezivanja zahtjeva na sustav s testnim slučajevima. Na slici 2.5 prikazano je sučelje softvera *TestCollab*.[12] Jedna od glavnih zamjerki mu je ta što se prilikom sačinjavanja izvještaja mora primjenjivati jedan od njegovih predložaka i ne postoji mogućnost kreiranja vlastitog predloška za izvještaje.[13]



Sl. 2.5. Prikaz izgleda sučelja *TestCollab* softvera.

2.2. RAZLOZI RAZVIJANJA VLASTITOG RJEŠENJA

Naizgled, možemo reći da je većina postojećih rješenja kvalitetna i dobra.

Kada je u pitanju softver koji se razvija u ovom diplomskom radu, svi navedi komercijalni programi u potpoglavlju 2.1 bi i više nego dobro zadovoljili sve potrebe navedene u zahtjevima, a i sve potrebe nekih većih tvrtki koje se bave testiranjem. Ipak, skoro sve tvrtke koje se bave isključivo testiranjem ili koje kroz vlastiti razvoj rade puno testova(vremenski i količinski), ne koriste se ovakvim alatima nego razvijaju vlastiti interni alat. Logično pitanje koje se postavlja je - zašto?

Alati koji su navedeni u potpoglavlju 2.1 su po autorovim spoznajama najkvalitetniji komercijalni alati za upravljanje testovima, što je područje teme ovog diplomskog rada. Nijedan od spomenutih alata nije besplatan niti je otvorenog koda. Prosječna najniža cijena je između 15 i 39 američkih dolara po jednom korisniku mjesečno. To je jedan od odgovora na pitanje „zašto se razvijaju ovakvi alati“. Drugi i važniji razlog je taj što se interno razvijenom alatu može pristupiti putem izvornog koda, prepraviti ga ovisno o potrebama i u potpunosti prilagoditi da bi se testovi mogli što efikasnije izvršavati. Također, što je najvažnije, takav alat kojem se može pregledati izvorni kod jer je interno razvijan, zadovoljava sigurnosne zahtjeve tvrtki i korporacija, jer se točno zna na što se alat povezuje, s čime i kime komunicira, te se može uspostaviti i za lokalnu upotrebu.

Ovim diplomskim radom razvijeno je primarno i pokazno rješenje. Primarno rješenje je izrađeno kroz rad u tvrtki ***TTTech-Auto*** i dosta je opširnije i kompleksnije od pokaznog. Zbog izjave o zaštiti privatnosti, te nemogućnosti prezentiranja originalnog rješenja, napravljeno je pokazno rješenje, a koje je elaborirano u ovom radu. Zamisao i svrha primarnog softverskog rješenja je direktna integracija na postojeći interni sustav kvarova tvrtke (*JIRA*) i sustav u kojem se kreiraju testne kampanje i testni ciljevi (*Winshell*), tako da se za izvedene testove podaci povuku iz baze i takvi obrađuju, izvoze i prikazuju - s ciljem optimiziranja vremena trajanja izvođenja testova, vremena zauzeća resursa i vremena zauzeća testera. U pokaznom rješenju zbog nemogućnosti spajanja na internu bazu tvrtke i pristup testovima, kreirano je generiranje nasumičnih testnih kampanja i obrada tako generiranih rezultata, kako bi se dobio jasan uvid u osnovnu funkcionalnost softverskog rješenja.

3. PROJEKTIRANJE SUSTAVA

U ovom poglavlju pružen je uvid u zahtjeve postavljene na sustav, kao i opis rješenja prije njegove implementacije.

3.1. ZAHTJEVI NA SUSTAVU

Programska podrška za aplikaciju treba biti implementirana kroz mrežnu aplikaciju. Potrebno je grafički prikazati podatke spremljene u bazu podataka. Koncept baze podataka, tablica i polja u istoj potrebno je prilagoditi potrebama. Za svaki izvršeni test potrebno je imati spremljeno vrijeme početka, kraja i trajanja izvršenja testa.

Potrebni podatci i njihovi opisi koji se spremaju za test, prikazani su na tablici 3.1.

Tab. 3.1. Opisi podataka koji su potrebni za pojedini test.

NAZIV POLJA	OPIS
naziv	naziv testa
vrsta izvršenja (planirana)	način na koji se test planira izvršiti (ručno ili automatski)
kampanja	testna kampanja kojoj test pripada
status testa	status izvršenja testa (izvršen ili neizvršen)
prosječno trajanje	planirano trajanje shodno prethodnim izvršenjima istog testa
tester	osoba koja je izvršila testiranje
ploča	ploča ili radna stanica na kojoj se test izvršio
smjena	radna smjena u kojoj je test izvršen
početak izvršenja	datum i vrijeme početka testiranja
kraj izvršenja	datum i vrijeme kraja testiranja
trajanje izvršenja	ukupno vrijeme trajanja testiranja
status izvršenja	status izvršenog testa u obliku rezultata (prošao ili pao)

Za statistiku o izvršenim testovima i za pripadajuće grafičke prikaze potrebno je omogućiti izvoz iz sustava u obliku izvještaja, tj. dokumenta u obliku zapisa PDF ili DOCX. Izvoz podataka o testovima treba omogućiti u oblicima zapisa XLSX, JSON, CSV i XML.

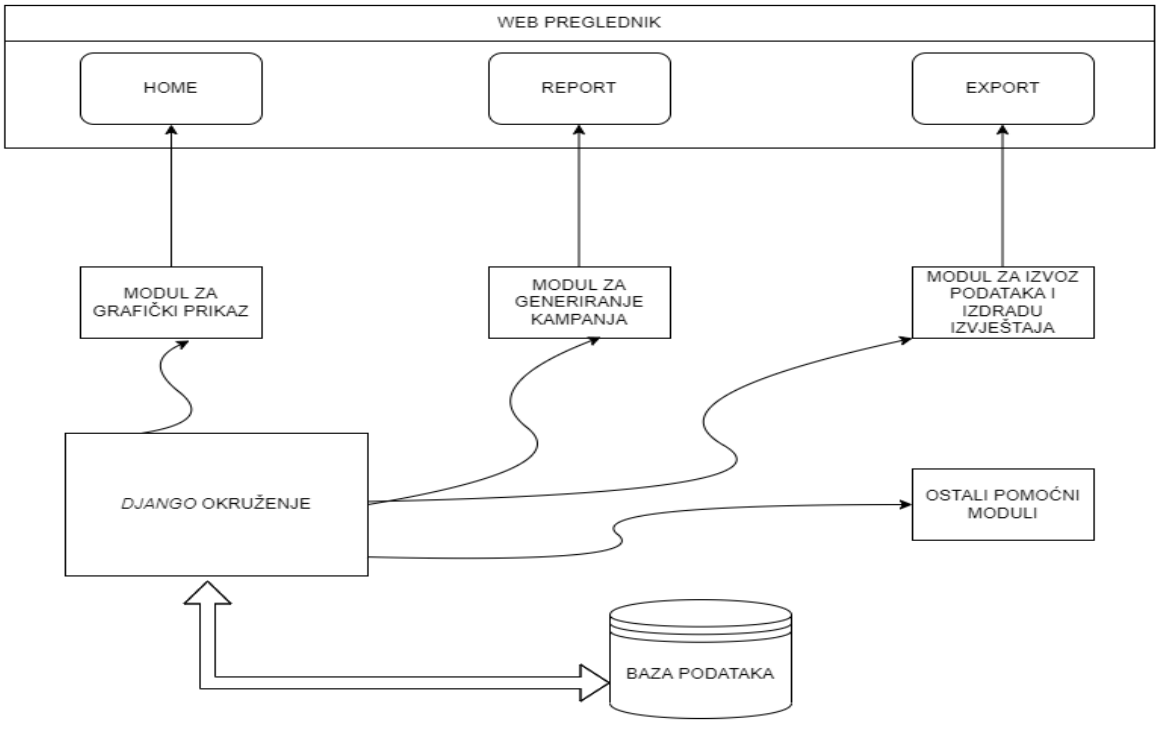
3.2. PROJEKTNO RJEŠENJE

U ovom potpoglavlju prikazani su dijagrami prijedloga rješenja izgleda i funkcionalnosti sustava. Razumijevajući zahtjeve, sustav je osmišljen da svoju funkciju obavlja kroz tri glavna modula, odnosno kroz tri glavna ekrana unutar internetske aplikacije:

- ***Home*** – početna stranica, zadužena za grafički prikaz rezultata direktno u internetskom sučelju
- ***Manage*** – druga stranica, zadužena za dodavanje novih kampanja, njihovo izvršenje i brisanje
- ***Report*** – treća stranica, zadužena za površni prikaz kampanja u sustavu, izradu izvještaja i izvoz podataka.

3.2.1. FUNKCIJSKA ARHITEKTURA SUSTAVA

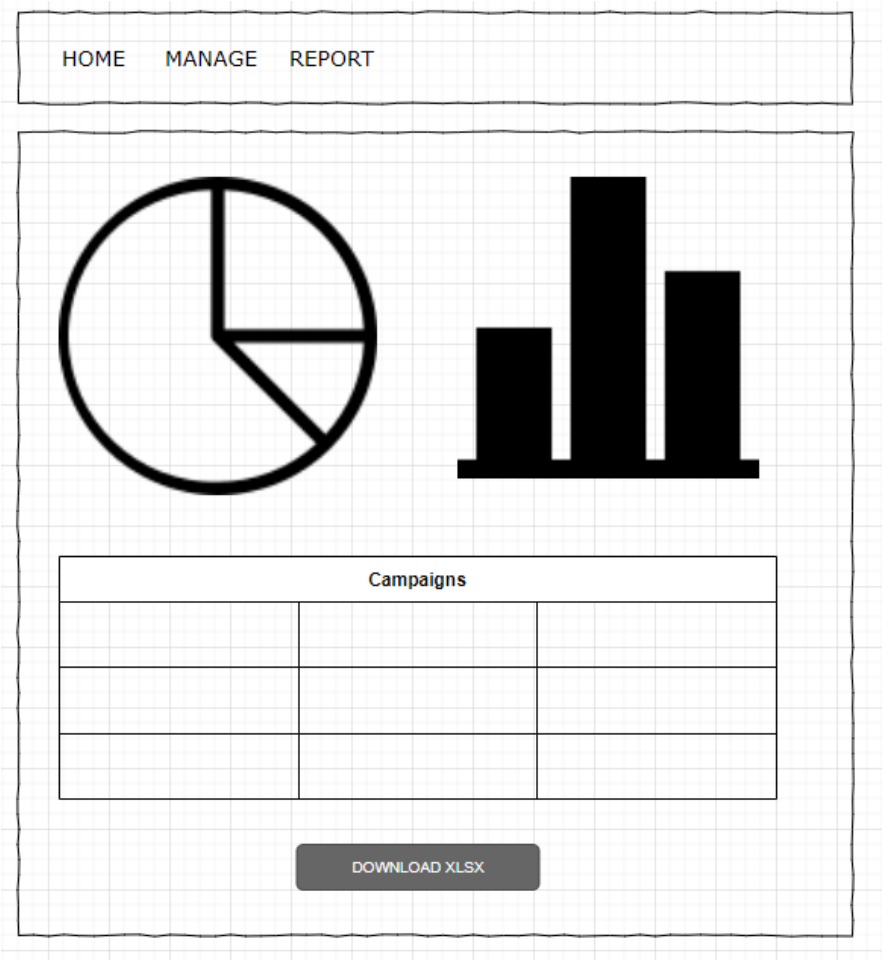
U ovom potpoglavlju prikazana je arhitektura sustava putem dijagrama. (Slika 3.1.) Sva programska funkcionalnost sustava nalazi se u okruženju *django*.



Sl. 3.1. Funkcijski dijagram arhitekture sustava.

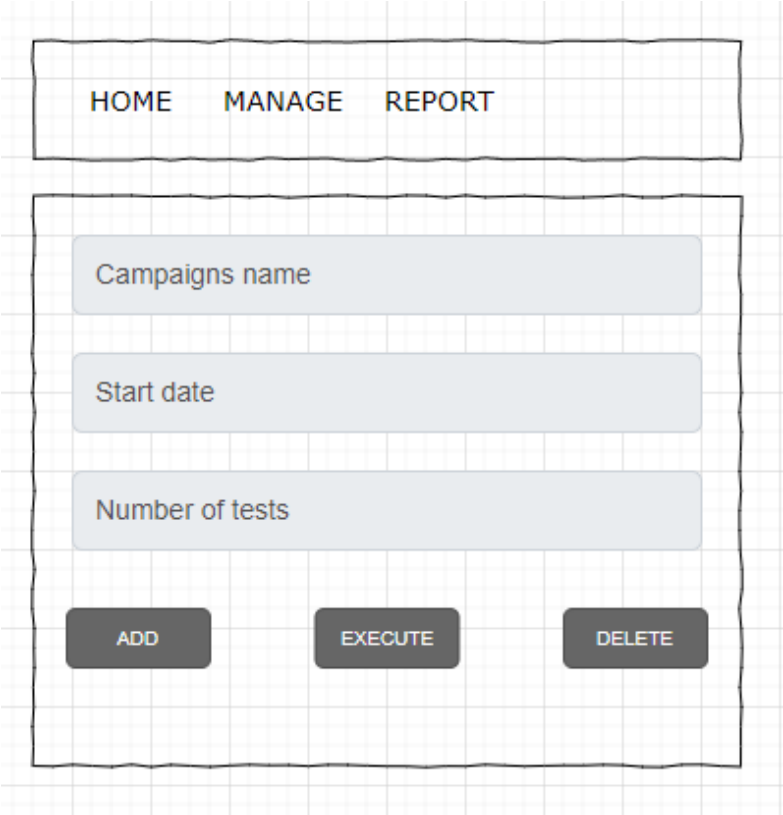
3.2.2. IDEJNI IZGLED SUČELJA

Izgled početne, *Home* stranice, prikazan je na slici 3.2. Na *Home* stranici nalaze se tri grafička elementa: kružni graf, stupčasti graf i podatkovna tablica. Na kružnom grafu prikazuju se odnosi uspješno i neuspješno provedenih testova.



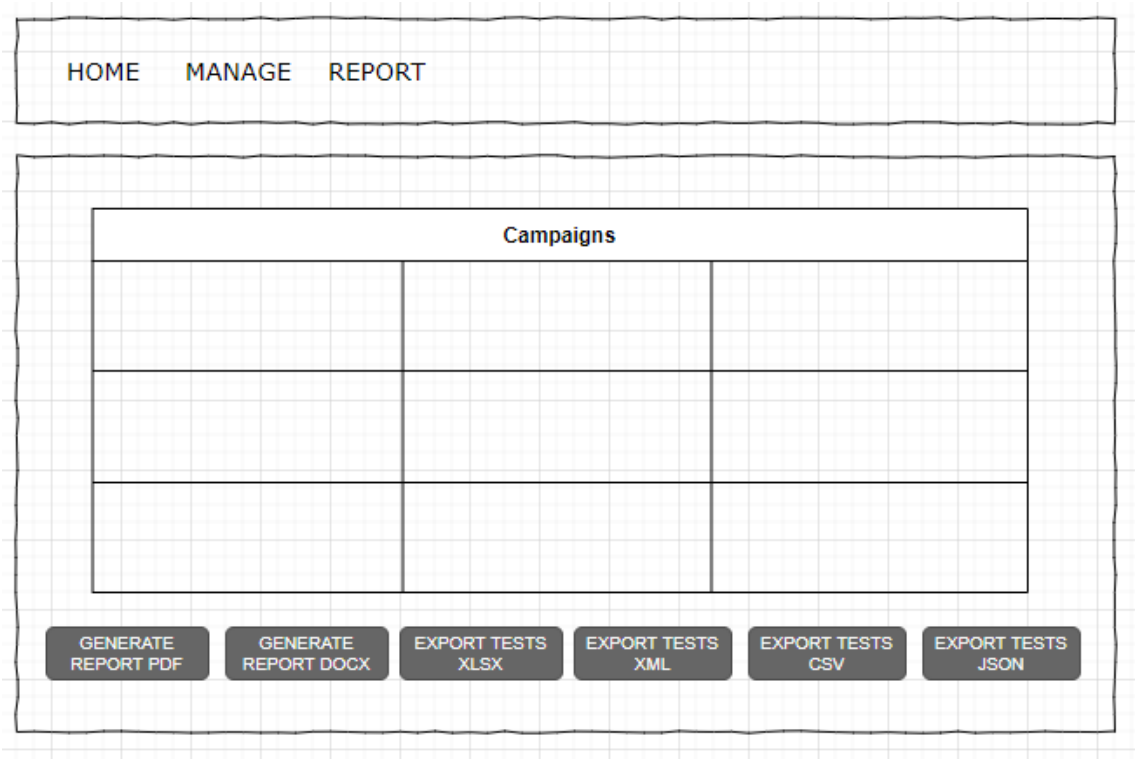
Sl. 3.2. Skica rasporeda (engl. *layout*) *Home* prikaza.

Na slici 3.3 prikazana je skica idejnog izgleda stranice *Manage*. Na ovoj stranici korisnik u polja unosi podatke o kampanji, koju potom kreira, izvršava ili briše - pritiskom na jednu od tri tipke.



Sl. 3.3. Dijagram rasporeda (engl. *layout*) *Manage* prikaza.

Na slici 3.4. prikazana je skica izgleda stranice *Report*. Na ovoj stranici kroz podatkovnu tablicu ispisane su sve kreirane kampanje. Pritiskom na neku od tipki, generiraju se izvještaji, ovisno o tipki koja je pritisnuta.



Sl. 3.4. Dijagram rasporeda (engl. *layout*) *Export* prikaza.

4. RAZVOJ PROGRAMSKOG SUČELJA

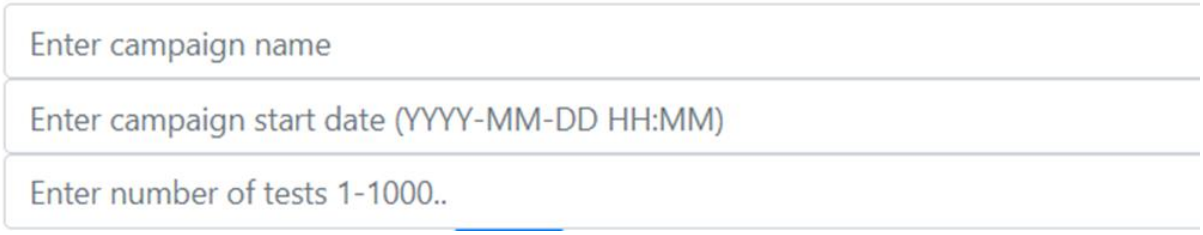
U ovom poglavlju kroz tri potpoglavlja objašnjena je implementacija triju glavnih modula putem kojih se kontrolira čitav sustav. To su moduli koji su idejno nacrtani u funkcijskom dijagramu sustava prikazanom u prethodnom poglavlju, na slici 3.1.

4.1. MODUL ZA GENERIRANJE TESTNIH KAMPANJA

Radi lakšeg testiranja i razvoja aplikacije kroz jedan glavni modul, odnosno klasu u *python* programskom jeziku *ManageScreen.py*, kreirano je više modula koji međusobno komuniciraju i generiraju nasumične podatke o testnoj kampanji i testovima, ovisno o ulaznim parametrima koji su ručno uneseni kroz internetsko sučelje aplikacije. Na slici 4.1 može se vidjeti izgled *Manage* prikaza za unos podataka o kampanji.

Sl. 4.1. Izgled *Manage* prikaza.

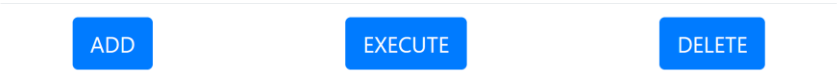
Prema slici 4.2, generiranje kampanje i testova obavlja se tako da korisnik popuni tri tražena polja.



Sl. 4.2. Polja za kreiranje kampanje s *Manage* prikaza.

U prvom je polju naziv kampanje; u drugom datum i vrijeme početka kampanje, a što se unosi u obliku zapisa *YYYY-MM-DD HH:MM* - gdje *YYYY* predstavlja godinu, *MM* mjesec, *DD* dan, *HH* sat i *MM* minute početka izvršenja kampanje. U treće polje unosi se broj testova, a trenutno je u aplikaciji ograničen broj testova na maksimalno 1000, radi lakšeg testiranja. Broj testova može biti i puno veći ako server na kojem se nalazi baza podataka to može podržati.

Nakon unosa podataka, potrebno je pritisnuti tipku *ADD* (Slika 4.3.).



Sl. 4.3. Tipke s *Manage* prikaza za upravljanje kampanjom.

Pritiskom tipke *ADD*, uz pomoć instance klase *DataBaseHandler.py* poziva se metoda *generateCampaign(campaignName, campaignStartData, testCount)*, koja prima tri navedena podatka kao argumente. Metoda *generateCampaign()* prvo napravi objekt klase modela *Campaign*, koji predstavlja jedan zapis u tablici *Campaign*, baze podataka *SqlLite*. Izgled definicije klase *Campaign* prikazan je na slici 4.4 i predstavlja tablicu *Campaign* u programskom okruženju *python*.

```
class Campaign(models.Model):
    id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=50)
    start_date = models.DateTimeField()
```

Sl. 4.4. Programski kôd klase modela *Campaign.*

Na slici 4.4 vidljivo je da tablica *Campaign* ima samo tri atributa, odnosno polja, primarni ključ (engl. *primary key*), ime i datum početka. Na slici 4.5. prikazan je primjer iz programskog koda kreiranja objekta.

```
newCampaign=Campaign(
    name=campaignName,
    start_date=startDate
)
newCampaign.save()
```

Sl. 4.5. Programski kod dijela funkcije *generateCampaign()* za spremanje kampanje u bazu.

Kreirani objekt sprema se u bazu podataka tek prilikom poziva metode *save()* na taj objekt klase *Campaign*.

U nastavku izvođenja pozvane metode *generateCampaign()* kreiraju se objekti klase modela *Test* - onoliko puta koliko je zadanih testova. Definicija klase *Test* prikazana je na slici 4.6.

```
class Test(models.Model):
    id = models.AutoField(primary_key=True)
    campaign = models.ForeignKey(Campaign, models.CASCADE)
    status = models.CharField(max_length=11, choices=PROGRESS_STATE,
                              default='unexecuted')
    name = models.TextField(blank=True, null=True)
    average_duration = models.TimeField(blank=True, null=True)
    runtype = models.CharField(max_length=11, choices=EXECUTION_TYPE,
                              default='manual')
    execution_report = models.ForeignKey(TestExecutionReport, models.DO_NOTHING,
                                         blank=True, null=True)
```

Sl. 4.6. Programski kôd klase modela *Test*.

Tablica *Test* ima sedam polja, odnosno prikaz tablice u programskom kodu klasa *Test*, ima sedam atributa koji su opisani u tablici 4.1.

Tab. 4.1. Atributi i njihovi opisi klase modela *Test.*

NAZIV	OPIS
id	jedinstveni identifikacijski broj zapisa u tablici
campaign	strani ključ na tablicu *Campaign* (nužno za kreiranje zapisa)
status	trenutni status testa, prilikom kreiranja svi su testovi u statusu neizvršeni (engl. *unexecuted*)
name	opis/ime testa
average_duration	procijenjeno trajanje izvršenja testa, procjena trajanja za testove se radi na osnovi prethodnih izvođenja istog testa
runtype	pretpostavljeni način izvođenja testova može biti ručni (engl. *manual*) ili automatski (engl. *automated*)
execution_report	strani ključ na tablicu *TestExecutionReport* prilikom kreiranja zapisa testa u bazu, ovo polje je prazno. Kada se test izvrši, u bazi se kreira zapis tablice *TestExecutionReport* te se njegov primarni ključ poveže na strani ključ ovog polja u tablici *Test*

Izgled jedne iteracije kreiranja objekta klase *Test* i njegovo spremanje, prikazano je na slici 4.7.

```
newTest=Test(
      name="blank",
      campaign=newCampaign,
      runtype=runType,
      average_duration=str(timedelta(minutes=random.randrange(5, 15)))
      )
newTest.save()
```

Sl. 4.7. Programski kôd dijela funkcije *generateCampaign()* za spremanje testa u bazu.

Tablica *TestExecutionReport* predstavljena je klasom modela *TestExecutionReport* na slici 4.8.

```
class TestExecutionReport(models.Model):
    id = models.AutoField(primary_key=True)
    board = models.ForeignKey(Board, models.CASCADE, blank=True, null=True)
    tester = models.ForeignKey(Tester, models.CASCADE)
    shift = models.ForeignKey(Shift, models.CASCADE)
    runtype = models.CharField(max_length=11, choices=EXECUTION_TYPE,
                               default='manual')
    start = models.DateTimeField(blank=True, null=True)
    end = models.DateTimeField(blank=True, null=True)
    duration = models.TimeField(blank=True, null=True)
    status = models.CharField(max_length=11, choices=EXECUTION_STATUS,
                               blank=True, null=True)
```

Sl. 4.8. Programski kôd klase modela *TestExecutionReport.*

Tablica ima devet polja reprezentiranih s atributima klase, u tablici 4.2 prikazani su nazivi atributa i njihovi opisi.

Tab. 4.2. Atributi i njihovi opisi klase modela *TestExecutionReport.*

NAZIV	OPIS
id	jedinstveni identifikacijski broj zapisa u tablici
board	strani ključ na tablicu *Board*, koja predstavlja radnu ploču/stanicu na kojoj se testovi izvršavaju
tester	strani ključ na tablicu *Tester*, koja predstavlja osobu koja izvršava test
shift	način na koji se test izvršio može biti ručni (engl. *manual*) ili automatski (engl. *automated*)
runtype	način na koji se test izvršio može biti ručni (engl. *manual*) ili automatski (engl. *automated*)
start	datum i vrijeme početka izvršenja testa
end	datum i vrijeme završetka izvršenja testa
duration	trajanje izvršenja testa
status	status rezultata izvedenog testa može biti: prošao (engl. *passed*) ili pao (engl. *failed*)

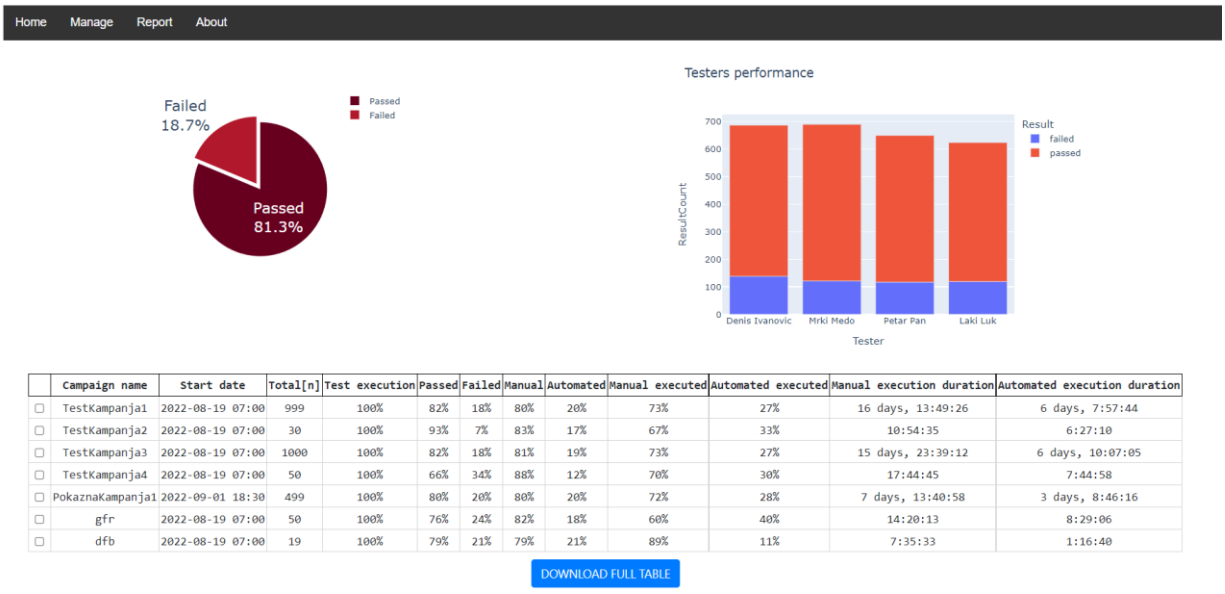
Prethodno opisana tablica popunjava se tek prilikom izvršenja testa, u ovom slučaju prilikom pritiska tipke *Execute* (prema slici 4.3), koji preko instance klase *DatabaseHandler* poziva metodu *executeCampaign(campaignName),* a koja automatski obavlja „izvršavanje“ testova i spremanje rezultata u bazu, tako da se obraća pažnja na tri ključna faktora izvršenja testa, a to su: tester, testna ploča i smjena. Ova su tri faktora bitna prilikom dodjeljivanja testova, a i njihovog izvršavanja da ne bi bilo preklapanja, jer jedan tester u pravilu radi samo u jednoj smjeni u danu. Također, u jednoj smjeni jednom testeru može biti dodijeljena samo jedan ploča, jer je pretpostavka da tester u isto vrijeme može raditi samo na jednoj ploči. Metoda *executeCampaign()* prima parametar *campaignName* iz polja za unos imena kampanje (prema slici 4.2) te se osnovi upisanog imena kampanje bazi podataka šalje upit za dohvaćanje zapisa o kampanji s tim imenom - ako postoji. Kada je dohvaćen zapis o kampanji i kada imamo primarni ključ iste, dohvaćaju se svi zapisi tablice *Test* koji pripadaju spomenutoj kampanji, odnosno čije polje *campaign* ima strani ključ kampanje za koje je korisnik unio ime.

Za svaki tako dohvaćeni test obavlja se „izvršavanje“ testa upisivanjem generiranih podataka u tablicu *TestExecutionReport* nakon čega se primarni ključ tog zapisa unosi u polje *execution_report*, koje u tablici *Test* predstavlja strani ključ na tablicu *TestExecutionReport* (Slika 4.6).

Zadnja tipka na slici 4.3 je *Delete,* koji poziva metodu *deleteCampaign(campaignName)* i za parametar joj predaje naziv kampanje iz polja za unos iste (Slika 4.2). Metoda *deleteCampaign()* u bazi, na osnovi predanog joj imena kampanje, dohvaća zapis o istoj te uz pomoć primarnog ključa tablice *Campaign,* dohvaća sve testove koji su u sastavu te kampanje. Potom iterira kroz sve testove i za svaki od njih, preko stranog ključa tablice *TestExecutionReport* (koji se nalazi u polju *execution_report*) dohvaća povezani zapis ove tablice. Na taj zapis/objekt tablice *TestExecutionReport* poziva se metoda *delete()* koja ga briše iz baze, a ista metoda se poziva i na nadređeni objekt/zapis tablice *Test* kako bi se i on izbrisao. Kada se na ovaj način obrišu svi zapisi o testovima i svi izvještaji o izvršenju testova za jednu kampanju, naposljetku se iz baze na isti način briše i sama kampanja, što završava postupak brisanja kampanje i povezanih tablica iz baze.

4.2. MODUL ZA GRAFIČKI PRIKAZ REZULTATA TESTOVA I PODATAKA O TESTOVIMA

U ovom potpoglavlju pokazano je softversko rješenje grafičkog i tabličnog prikaza testova i njihovih rezultata. Prikaz internetskog preglednika na početnoj stranici (engl. *Home*) vidljiv je na slici 4.9.



	Campaign name	Start date	Total[n]	Test execution	Passed	Failed	Manual	Automated	Manual executed	Automated executed	Manual execution duration	Automated execution duration
☐	TestKampanja1	2022-08-19 07:00	999	100%	82%	18%	80%	20%	73%	27%	16 days, 13:49:26	6 days, 7:57:44
☐	TestKampanja2	2022-08-19 07:00	30	100%	93%	7%	83%	17%	67%	33%	10:54:35	6:27:10
☐	TestKampanja3	2022-08-19 07:00	1000	100%	82%	18%	81%	19%	73%	27%	15 days, 23:39:12	6 days, 10:07:05
☐	TestKampanja4	2022-08-19 07:00	50	100%	66%	34%	88%	12%	70%	30%	17:44:45	7:44:58
☐	PokaznaKampanja1	2022-09-01 18:30	499	100%	80%	20%	80%	20%	72%	28%	7 days, 13:40:58	3 days, 8:46:16
☐	gfr	2022-08-19 07:00	50	100%	76%	24%	82%	18%	60%	40%	14:20:13	8:29:06
☐	dfb	2022-08-19 07:00	19	100%	79%	21%	79%	21%	89%	11%	7:35:33	1:16:40

Sl. 4.9. Izgled prikaza *Home*.

Na početnoj se stranici nalaze četiri komponente:

- kružni graf (engl. *pie chart*)
- stupčasti graf (engl. *bar chart*)
- tablica (engl. *table*)
- tipka (engl. *button*).

Sve su komponente napravljene uz pomoć biblioteke *dash*, odnosno pomoćne *dash*-biblioteke, *dash-bootstrap-components*.

Sav izgled i funkcionalnost početne stranice proizlazi iz *python*-modula *InfoScreen.py*. Unutar modula nalazi se klasa *InfoScreen*. Metoda *loadScreen()*, koja se nalazi unutar klase *InfoScreen*, uz pomoć biblioteka *channels* i *django-plotly-dash*, putem tablice iz baze podataka *django plotly dash* i putem klase *DjangoDash* pronalazi *div* komponentu, kojoj je atribut *ID* postavljen na vrijednost *HomeScreen*, a koju predajemo u konstruktoru klase (Slika 4.10).

```
def loadScreen(self):
    self.app=DjangoDash("HomeScreen",
                            external_stylesheets=[dbc.themes.BOOTSTRAP]
              )
    self.app.layout=html.Div()
    self.campaignsDataFrame = pd.DataFrame()
    self.appendContent(
        self.formatDataForTable(
            self.databaseHandler.getAllCampaigns()
        )
    )
```

Sl. 4.10. Programski kôd metode *loadScreen()* iz klase *InfoScreen*.

Predefinirana *div* komponente nalazi se u HTML-datoteci početne stranice (Slika 4.11). Nakon toga se sve ostale komponente koje generira programski kôd upisuju u prethodno spomenutu *div* komponentu.

```
{% load plotly_dash %}
<div id="HomeScreen" class="{% plotly_class name='HomeScreen' %} card"
                        style="height: 100%; width: 100%">
      {% plotly_app name='HomeScreen' ratio=0.59 %}
</div>
```

Sl. 4.11. Programski kôd predefinirane komponente *div* u HTML-datoteci početne stranice.

Kompletan izgled aplikacije upisuje se u varijablu *layout*, instance klase *DjangoDash*, komponentama biblioteke *DashBootstrapComponents* (Slika 4.12).

```
self.app.layout=html.Div(
            DCH.putInRows(
                [
                    DCH.getDownload("download-dataframe"),
                    DCH.putInCols([
                        DCH.getGraph_InfoScreen("pie_graph",
                                                PieChartHandler().getFigure()
                                               ),
                        DCH.getGraph_InfoScreen("bar_graph",
                                                BarChartHandler().getFigure()
                                               )
                    ]),
                    DCH.putInSpinner([
                        DCH.getDataTable_InfoScreen("campaignInfoTable",
                                                    dataForTable),
                    ]),
                    DCH.putInCols([
                        DCH.getButton_InfoScreen(
                                    "create_complete_campaigns_summary",
                                     "DOWNLOAD FULL TABLE"),
                    ])
                ]
            ),
            style={'width': '95%', 'padding': '0px'}
        )
```

Sl. 4.12. Programski kod dijela *appendContent()* metode iz klase *InfoScreen*.

DCH() kratica s prethodne slike predstavlja modul *DashComponentsHandler.py*, koji je proceduralan. To znači da nema klasa nego sadrži samo funkcije koje se pozivaju u programskom kôdu. U DCH-modulu nalaze se funkcije koje služe za generiranje *DashBootstrapComponents* komponenata.

Neke od funkcija su: *putInLoading(), getDataTableCampagns(), getInput(), getButton(), getDataTable(), putInCols(), getCol(), putInRows(), getGraph(), getDownload(),putInSpinner()*.

Metoda *appendContent(data)*, koja se poziva u metodi *loadScreen()*, za parametar prima objekt klase *DataFrame* iz biblioteke *Pandas*. Objekt *DataFrame* sadrži formatirane izvještaje o svim kampanjama. Tako formatiran objekt klase *DataFrame*, dobiva se kao odgovor na poziv funkcije *formatDataForTable(allCampaigns)*, koji za parametar prima listu objekata klase *Campaign*, a koji predstavljaju zapise tablice *Campaign* iz baze podataka. Funkcija *formatDataForTable()* iterativno za svaku kampanju poziva funkciju *getCampaignReportData(campaignID)* klase *DatabaseHandler*, koja prolazi kroz sve testove i njihove izvještaje za neku kampanju,

proračunava podatke i sumira rezultate. Funkcija *getCampaignReportData(campaignID)* vraća podatke u obliku objekta JSON, koji se onda u funkciji *formatDataForTable()* sortiraju i formatiraju za kreiranje liste *DataFrame*.

Polja koja se proračunavaju i formatiraju za svaku kampanju prikazana su u tablici 4.3.

Tab. 4.3. Atributi i njihovi opisi klase modela *TestExecutionReport.*

NAZIV	**OPIS**
CampaignName	ime kampanje
StartDate	datum i vrijeme početka izvršavanja kampanje
TotalTests	ukupan broj testova u kampanji
ExecutionProcent	postotak testova koji su izvršeni
PassedProcent	postotak testova koji su uspješno izvršeni
FailedProcent	postotak testova koji su neuspješno izvršeni
ManualProcent	postotak testova koji su namijenjeni za ručno izvršavanje
AutomatedProcent	postotak testova koji su namijenjeni za automatsko izvršavanje
ManualExecutedProcent	postotak testova koji su izvršeni ručno
AutomatedExecutedProcent	postotak testova koji su izvršeni automatski
ManualExAutomatedProcent	postotak testova koji su namijenjeni za ručno izvršavanje, a izvršeni su automatski
AutomatedExManualProcent	postotak testova koji su namijenjeni za automatsko izvršavanje, a izvršeni su ručno
ManualExManualProcent	postotak testova koji su namijenjeni za ručno izvršavanje, a izvršeni su ručno
AutomatedExAutomatedProcent	postotak testova koji su namijenjeni za automatsko izvršavanje a izvršeni su automatski
ManualExecutionDuration	ukupno vrijeme trajanja izvršenja svih ručnih testova
AutomatedExecutionDuration	ukupno vrijeme trajanja izvršenja svih automatskih testova
FirstShiftBoardUsage	broj različitih ploča koje su korištene u prvoj smjeni
FirstShiftTesterUsage	broj različitih testera koji su testirali u prvoj smjeni
FirstShiftTestsExecuted	ukupan broj testova koji su izvršeni u prvoj smjeni
SecondShiftBoardUsage	broj različitih ploča koje su korištene u drugoj smjeni
SecondShiftTesterUsage	broj različitih testera koji su testirali u drugoj smjeni
SecondShiftTestsExecuted	ukupan broj testova koji su izvršeni u drugoj smjeni

Neka od navedenih polja iz tablice 4.3 prikazana su u tablici na početnom prikazu (Slika 4.13).

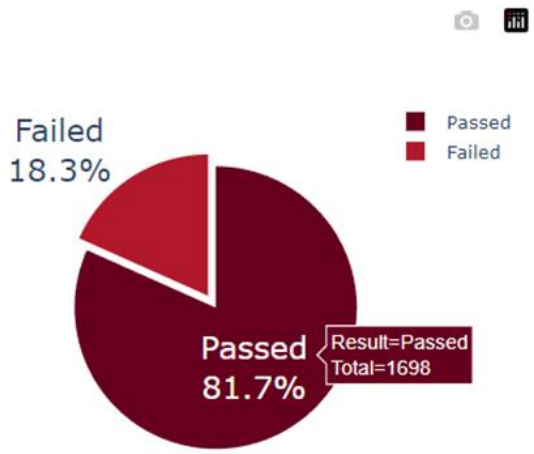
	Campaign name	Start date	Total[n]	Test execution	Passed	Failed	Manual	Automated	Manual executed	Automated executed	Manual execution duration	Automated execution duration
☐	TestKampanja1	2022-08-19 07:00	999	100%	82%	18%	80%	20%	73%	27%	16 days, 13:49:26	6 days, 7:57:44
☐	TestKampanja2	2022-08-19 07:00	30	100%	93%	7%	83%	17%	67%	33%	10:54:35	6:27:10
☐	TestKampanja3	2022-08-19 07:00	1000	100%	82%	18%	81%	19%	73%	27%	15 days, 23:39:12	6 days, 10:07:05
☐	TestKampanja4	2022-08-19 07:00	50	100%	66%	34%	88%	12%	70%	30%	17:44:45	7:44:58

DOWNLOAD FULL TABLE

Sl. 4.13. Izgled tablice s prikaza *Home*.

Pritiskom na tipku *DOWNLOAD FULL TABLE*, kompletni se podaci iz *DataFrame* liste pozivom metode *to_excel* prebacuju u *MS Excel* format datoteke, te započinje preuzimanje datoteke .XLSX na korisnikovo računalo.

Osim tablice s podacima, na stranici možemo dobiti informacije i iz grafova; kružni graf prikazuje odnos uspješno izvršenih testova i neuspješno izvršenih testova (Slika 4.14).



Sl. 4.14. Izgled kružnog grafa sa *Home* prikaza.

Konkretno u ovom slučaju vidi se da, od svih izvršenih testova, postotak uspješno izvršenih (engl. *passed*) iznosi 81,7 %, dok je 18,3 % testova neuspješno izvršeno (engl. *failed*). Također, prelaskom pokazivača preko jednog od dijelova kružnog grafa, pokazuje nam se lebdeći dijalog s imenom pripadajuće skupine rezultata, kao i broj testova s takvim ishodom. Ovaj graf napravljen je uz pomoć *python*-biblioteke *plotly*.

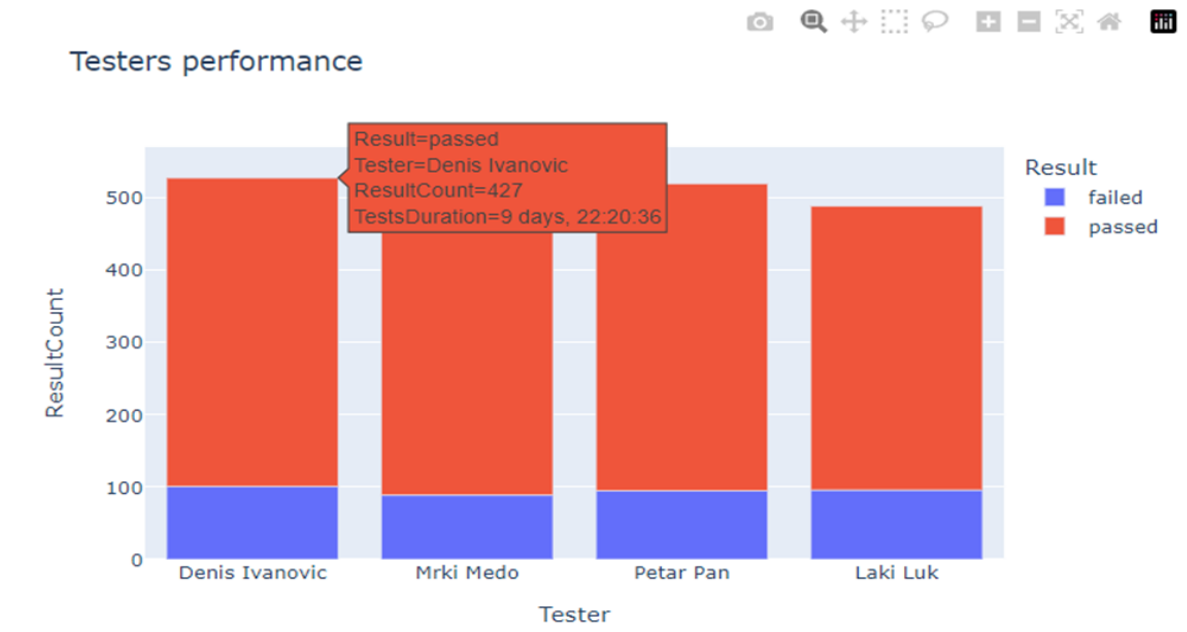
Sav posao vezan za kreiranje kružnog grafa te pripremu i obradu podataka za isti, obavlja se u modulu *PieChartHandler.py*. U modulu *InfoScreen.py* se samo poziva funkcija *getFigure()* klase *PieChartHandler*, koja vraća gotov prikaz grafa u obliku vektorske grafike *.svg*. Prikaz grafa se kreira unutar klase *PieChartHandler* u funkciji *createFigure(data)*, koja za parametar prima pripremljene podatke u obliku objekta klase *DataFrame* (Slika 4.15).

```
def creteFigure(self, data):
    figure = px.pie(
        data,
        values='Total',
        names='Result',
        color_discrete_sequence=px.colors.sequential.RdBu,
        width=400,
        height=400
        )
    figure.layout.autosize=False
    figure.update_traces(
        textfont_size=20,
        textinfo='label+percent',
        pull=[0.1, 0, 0.2, 0, 0, 0]
        )
    return figure
```

Sl. 4.15. Programski kôd funkcije *createFigure()*iz klase *PieChartHandle*.

Sirovi podaci o testovima dohvaćeni iz baze podataka uz pomoć funkcije *getAllTests()* se, prije predaje funkciji *createFigure()*, obrađuju unutar privatne funkcije *__getTestsPassedFailedData(allTests)*. Prema slici 4.15, vidljiv je način na koji se kreira prikaz grafa, odnosno koji parametri su postavljeni i kako. Funkcija *Pie()* prima više od dvadeset parametara putem kojih možemo prilagoditi kružni graf na način na koji je nama potreban. U ovom slučaju postavljeno ih je samo šest.

Drugi graf na početnoj stranici je stupčasti graf (Slika 4.16). Namjena mu je omogućiti detaljan uvid u učinak testera koji su uključeni u izvršenje testova.



Sl. 4.16. Izgled stupčastog grafa s prikaza *Home*.

Slično kao i kod kružnog grafa, sav posao pripreme podataka i iscrtavanja njegove figure, obavlja se u odvojenom modulu napravljenom za ovu vrstu grafa - *BarChartHandler.py*. U nadređenom modulu *InfoScreen.py* poziva se funkcija *getFigure()* klase *BarChartHandle*, koja kao odgovor vraća prikaz grafa u obliku vektorske grafike (*.svg*). Unutar funkcije *getFigure()* poziva se funkcija *createFigure()* (Slika 4.17), koja kao parametar prima objekt klase *DataFrame*, a koji predstavlja uređen popis informacija o učinku svakog testera.

```
    def createFigure(self, data):
        if not data.empty:
            figure = px.bar(
                        data,
                        x="Tester",
                        y="ResultCount",
                        color="Result",
                        title="Testers performance",
                        hover_data=['TestsDuration', 'Result',
'ResultCount']
                       )
            return figure
        else:
            return None
```

Sl. 4.17. Programski kod funkcije *createFigure()* iz klase *BarChartHandler*.

Takav uređeni *DataFrame* popis dobiva se pozivanjem privatne funkcije *BarChartHandler* klase *__getTestersPerformanceData()*. Funkcija *__getTestersPerformanceData()* uz pomoć instance klase *DatabaseHandler* iz baze dohvaća sve testere, iterativno prolazi kroz popis istih i za svakog dohvaća izvještaje o izvršenju testova, iz kojih onda dobiva informaciju o učinku svakog od njih.

Prema tome, na slici 4.16 za svakog se testera može vidjeti odnos testova koje su uspješno izvršili, kao i onih koje su izvršili neuspješno. Kada se pokazivačem pređe preko jednog od stupaca grafa, može se vidjeti koji rezultat on predstavlja, ime testera kojem taj rezultat pripada, broj testova s tim rezultatom (a izvršio ih je taj tester) i na kraju, ukupno vrijeme trajanja završetka tog broja testova.

4.3. MODUL ZA GENERIRANJE IZVJEŠTAJA I IZVOZ PODATAKA

U ovom potpoglavlju prikazano je softversko rješenje generiranja izvještaja i izvoza podataka iz baze u razne formate.

Izvještaj o kampanji moguće je generirati u dva formata:

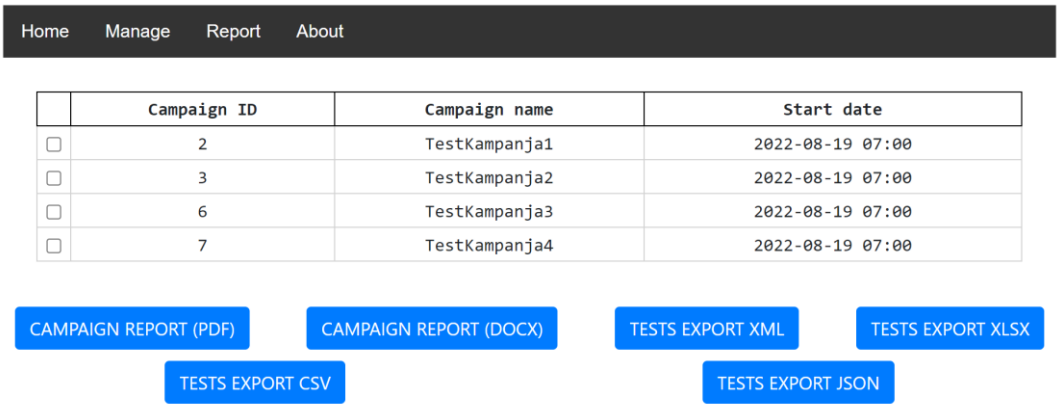
- *Portable Document Format* (PDF)
- *Microsoft Office Word Format* (DOCX)

Također se kao jedan oblik izvještaja može računati tablica s izvještajem svih kampanja koje se preuzimaju na početnoj stranici u *Microsoft Office Excel Formatu* (XLSX).

Podatke o izvršenim testovima moguće generirati u četiri formata:

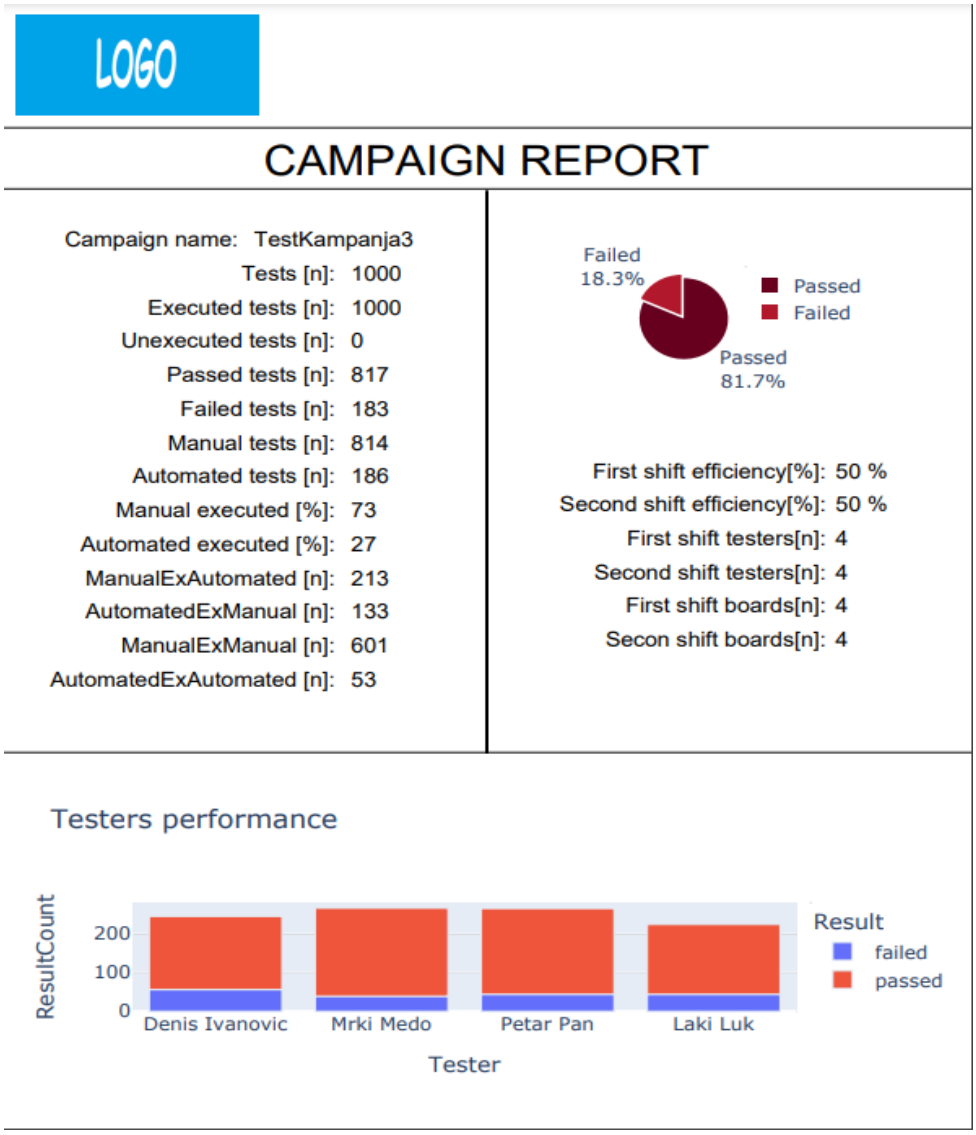
- *Extensible Markup Language* (XML)
- *Microsoft Office Excel* (XLSX)
- *JavaScript Object Notation* (JSON)
- *Comma-separated Values* (CSV).

Na slici 4.18 prikazan je izgled internetske stranice za generiranje izvještaja (engl. *Report*).



Sl. 4.18. Izgled prikaza *Report*.

Sav izgled i funkcionalnost internetske stranice za generiranje izvještaja i izvoz podataka proizlazi iz modula *ReportScreen.py*. Prilikom pritiska tipke *CAMPAIGN REPORT (PDF)* na instancu klase *ReportGenerator* modula *ReportGenerator.py* - poziva se funkcija *buildCampaignReport(campaignID, reportName)*, koja kao parametar prima primarni ključ kampanje za koju se izvještaj sačinjava, kao i željeno ime izvještaja. Kampanja se odabire tako da se u tablici (prema slici 4.18) pored željene kampanje označi okvir za izbor pritiskom pokazivača na isti (prvi stupac tablice). Modul *ReportGenerator.py* zadužen je za ručno iscrtavanje izvještaja i upis podataka u PDF-datoteku. Iscrtavanje i kreiranje PDF-datoteke radi se uz pomoć biblioteke *Canvas*. Zadani format dokumenta je A4. Kao pomoć klasi *ReportGenerator* primjenjuje se modul *ReportDimensionConstructor*, koji prima dva parametra; visinu i širinu dokumenta, te proračunava potrebne lokacije na dokumentu na koje će se ispisivati pojedini podaci i grafovi. Prvo se pozivom funkcije *getCampaignReportData(campaignID)* na instancu klase *DatabaseHandler* dohvaćaju podaci o izvještaju kampanje. Dohvaćeni podaci se potom upisuju i ucrtavaju u instancu klase *Canvas*. Kada se podaci uz pomoć više raznih metoda ucrtaju i upišu, dolazi do spremanja PDF-datoteke (Slika 4.19). Funkcija *buildCampaignReport()* potom vraća putanju do izvještaja, za koji onda počinje preuzimanje na korisnikovo računalo.



Sl. 4.19. Izgled PDF-izvještaja generiranog na prikazu *Report*.

Pritiskom na tipku *CAMPAIGN REPORT* (DOCX) kreira se izvještaj o kampanji u DOCX-formatu, istog izgleda kao i onaj u PDF-u, prikazan na slici 4.19. Programski se izvodi isti postupak kao i kod kreiranja PDF-izvještaja, osim što se kod kreiranja DOCX-izvještaja, nakon kreiranja PDF-izvještaja, isti pretvara u format DOCX. Pozivom konstruktora klase *Converter* modula *pdf2docx*, koji kao parametar u konstruktor prima putanju do datoteke PDF-dokumenta. Potom se na instancu klase *Converter* poziva funkcija *convert(filePathDOCX*, koja za parametar prima lokaciju na koju će se spremiti nova DOCX-datoteka te, kao i kod PDF-izvještaja, funkcija *buildCampaignReport(campaignID, reportName)* vraća putanju do datoteke novokreiranog DOCX-izvještaja za koji se pokreće preuzimanje na računalo korisnika.

Izvoz podataka u format XML izvodi se tako da se poziva metoda *createXMLFile(data, filePath)* iz klase *ReportScreen*, koja prima podatke u obliku *DataFramea* te ih ručno prebacuje u XML-

format. Prema slici 4.20, metoda *createXMLFile()* preuređuje *DataFrame* podatke u XML-format

```
def createXMLFile(self, data, filePath):
    xml_data = ['<root>']
    for column in data.columns:
        xml_data.append('<{}>'.format(column))
        for field in data.index:
            xml_data.append('<{0}>{1}</{0}>'.format(field,
                                data[column][field]))
        xml_data.append('</{}>'.format(column))
    xml_data.append('</root>')
```

Sl. 4.20. Programski kod *createXMLFile()* funkcije iz *ReportScreen* klase.

za koji se poslije kreira datoteka XML, koja se preuzima na korisnikovo računalo prilikom pritiska tipke *TESTS EXPORT XML*, na internetskoj stranici (Slika 4.18).

Dio programskog kôda (prema slici 4.21) unutar funkcije *callback* za tipke (Slika 4.18), koji se odnosi na radnju kada je pritisnuta tipka *TESTS EXPORT XLSX*.

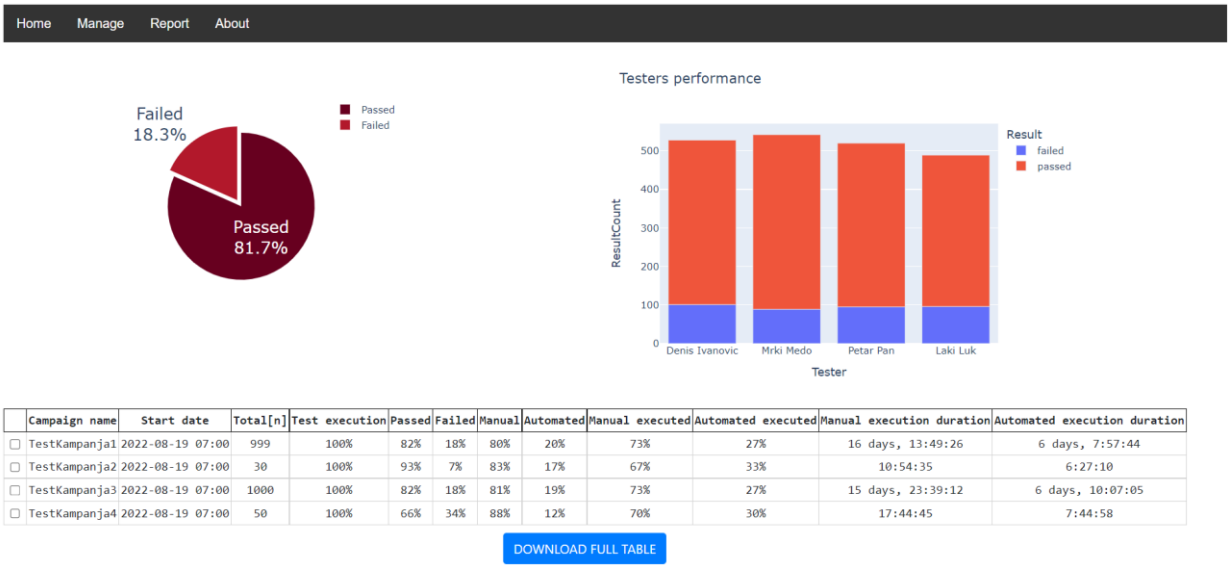
```
if createTestsExportToXLSX:
      for id in selectedRowsIDs:
      fileName=allData[id]["Name"]+"_Tests.xlsx"
      data=self.getTestsDataToExport(allData[id]["ID"])
      return dash.no_update, None, dash.no_update, dash.no_update,
dash.no_update, dash.no_update, dcc.send_data_frame(data.to_excel,
fileName, sheet_name="All_tests"), dash.no_update, dash.no_update, rows
```

Sl. 4.21. Programski kôd funkcije *callback* iz klase *ReportScreen*

Ovaj dio programskog kôda prikazuje postupak izvoza testova i izvještaja o njima za pojedinu kampanju u XLSX-formatu, a postupak je identičan za formate JSON i CSV. Prvo se kreira naziv datoteke, pa se uz pomoć funkcije *getTestsDataToExport()* dohvaćaju podaci o testovima, nakon čega se taj popis testova - koji je u *DataFrame* obliku - uz pomoć metode *send_data_frame()* iz *DashCoreComponents* klase, direktno šalje komponenti *download*, koja počinje preuzimanje datoteke na korisnikovo računalo.

5. PRIMJER RADA APLIKACIJE

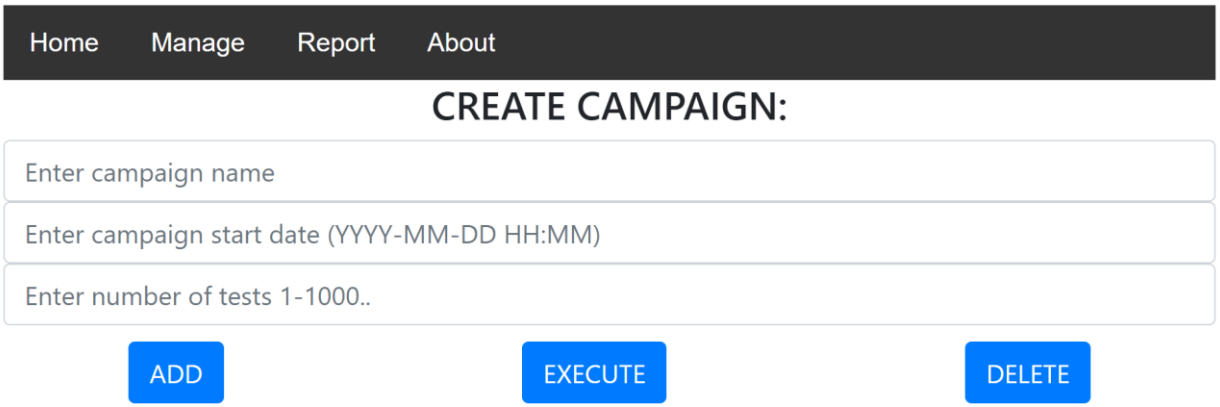
Korisnik nakon otvaranja internetske stranice ulazi na početnu stranicu *Home*, prikazanu na slici 5.1.



	Campaign name	Start date	Total[n]	Test execution	Passed	Failed	Manual	Automated	Manual executed	Automated executed	Manual execution duration	Automated execution duration
☐	TestKampanja1	2022-08-19 07:00	999	100%	82%	18%	80%	20%	73%	27%	16 days, 13:49:26	6 days, 7:57:44
☐	TestKampanja2	2022-08-19 07:00	30	100%	93%	7%	83%	17%	67%	33%	10:54:35	6:27:10
☐	TestKampanja3	2022-08-19 07:00	1000	100%	82%	18%	81%	19%	73%	27%	15 days, 23:39:12	6 days, 10:07:05
☐	TestKampanja4	2022-08-19 07:00	50	100%	66%	34%	88%	12%	70%	30%	17:44:45	7:44:58

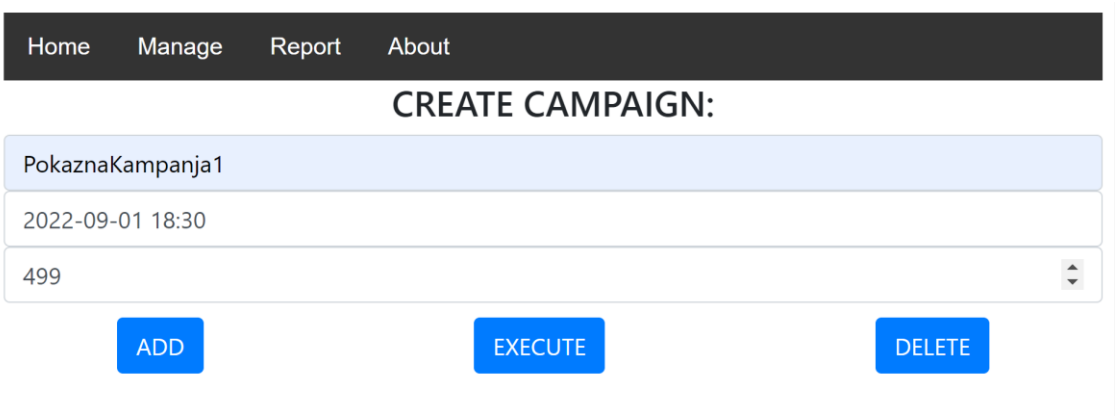
Sl. 5.1. Izgled početnog prikaza stranice *Home*.

Na početnom, *Home* prikazu, korisnik može vidjeti popis već dodanih kampanja, kao i dva grafa. Pritiskom na tipku *Manage* u gornjem lijevom uglu, otvara nam se prozor za upravljanje kampanjama, odnosno za njihovo dodavanje, izvršenje i brisanje (Slika 5.2).



Sl. 5.2. Izgled prikaza stranice *Manage*.

U stranici *Manage* možemo dodati kampanju ako unesemo tri tražena podatka. Na slici 5.3 prikazan je primjer ispravnog unosa kampanje.



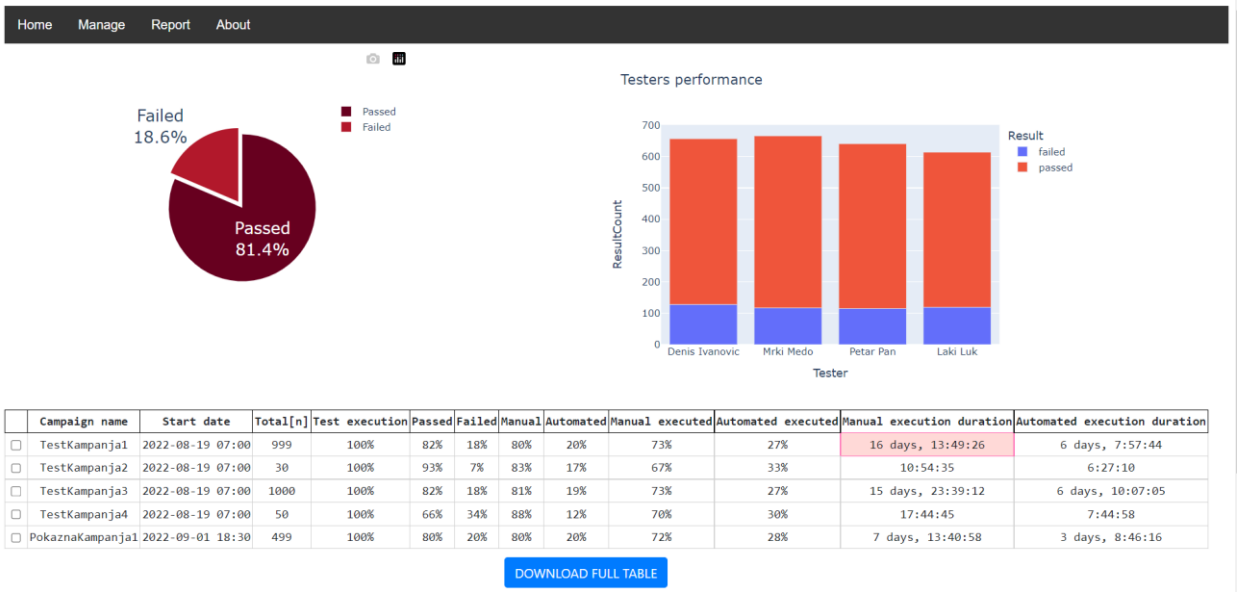
Sl. 5.3. Primjer ispravno unesene kampanje.

Kad su potrebni podatci za dodavanje kampanje uneseni, korisnik pritisne tipku *ADD* (prema slici 5.2), a potom se kampanja upiše u bazu podataka. Kada je kampanja upisana, ona nije izvršena i ako se korisnik vrati na početnu stranicu u tablici s popisom kampanja (Slika 5.4), vidjet će da je kampanja dodana, ali podatci o izvršenju će biti na nula posto.

	Campaign name	Start date	Total[n]	Test execution	Passed	Failed	Manual	Automated	Manual executed	Automated executed	Manual execution duration	Automated execution duration
☐	TestKampanja1	2022-08-19 07:00	999	100%	82%	18%	80%	20%	73%	27%	16 days, 13:49:26	6 days, 7:57:44
☐	TestKampanja2	2022-08-19 07:00	30	100%	93%	7%	83%	17%	67%	33%	10:54:35	6:27:10
☐	TestKampanja3	2022-08-19 07:00	1000	100%	82%	18%	81%	19%	73%	27%	15 days, 23:39:12	6 days, 10:07:05
☐	TestKampanja4	2022-08-19 07:00	50	100%	66%	34%	88%	12%	70%	30%	17:44:45	7:44:58
☐	PokaznaKampanja1	2022-09-01 18:30	499	0%	0%	0%	79%	21%	0%	0%	0:00:00	0:00:00

Sl. 5.4. Tablica s popisom kampanja prije izvršenja kampanje *PokaznaKampanja1*.

Da bi se kampanja izvršila, potrebno se vratiti na prikaz *Manage* i pritisnuti tipku *EXECUTE*. Tada se kampanja izvrši nasumično, a kada se ponovno uđe na početni prikaz, može se vidjeti ažuriran popis kampanja i grafova (Slika 5.5).

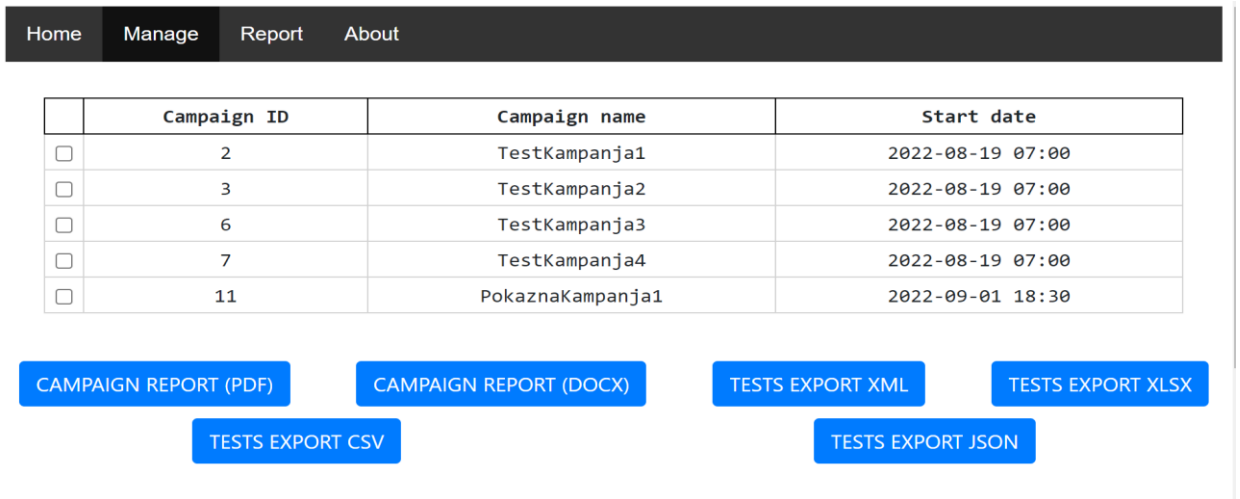


	Campaign name	Start date	Total[n]	Test execution	Passed	Failed	Manual	Automated	Manual executed	Automated executed	Manual execution duration	Automated execution duration
☐	TestKampanja1	2022-08-19 07:00	999	100%	82%	18%	80%	20%	73%	27%	16 days, 13:49:26	6 days, 7:57:44
☐	TestKampanja2	2022-08-19 07:00	30	100%	93%	7%	83%	17%	67%	33%	10:54:35	6:27:10
☐	TestKampanja3	2022-08-19 07:00	1000	100%	82%	18%	81%	19%	73%	27%	15 days, 23:39:12	6 days, 10:07:05
☐	TestKampanja4	2022-08-19 07:00	50	100%	66%	34%	88%	12%	70%	30%	17:44:45	7:44:58
☐	PokaznaKampanja1	2022-09-01 18:30	499	100%	80%	20%	80%	20%	72%	28%	7 days, 13:40:58	3 days, 8:46:16

Sl. 5.5. Izgled početnog prikaza *Home* stranice nakon izvršenja kampanje *PokaznaKampanja1*.

Kada je kampanja dodana i izvršena, ona se može na prikazu *Manage* i izbrisati ako se u polje za naziv kampanje unese točan naziv kampanje koju korisnik želi izbrisati. Pritiskom na tipku *DELETE*, ta se kampanja briše iz baze podataka.

Korisnik potom može pritisnuti tipku u gornjem lijevom uglu stranice *Report* i tako ući na treću stranicu, koja služi za izradu izvještaja o izvršenoj kampanji i za izvoz podataka (Slika 5.6).



Sl. 5.6. Izgled prikaza stranice *Report*.

Označavanjem kvadratića lijevo od *ID*-broja kampanje (prvi stupac tablice), na stranici *Report* može se odabrati jedna kampanja. Potom se pritisne jedna od šest tipki (prema slici 5.6), ovisno o tome što korisnik želi. U popisu prema tablici 5.1, možemo vidjeti koja tipka pokreće koju radnju.

Tab. 5.1. Radnje koje pokreću tipke s *Report* prikaza.

NAZIV TIPKE	RADNJA KOJU POKREĆE
CAMPAIGN REPORT (PDF)	Generira izvještaj o izvršenoj kampanji u PDF-formatu i preuzima ga na računalo korisnika
CAMPAIGN REPORT (DOCX)	Generira izvještaj o izvršenoj kampanji u DOCX-formatu i preuzima ga na računalo korisnika
TESTS EXPORT XML	Generira dokument u XML-formatu, sa svim testovima iz označene kampanje i s podatcima
TESTS EXPORT XLSX	Generira dokument u XLSX-formatu, sa svim testovima iz označene kampanje i s podatcima
TESTS EXPORT CSV	Generira dokument u CSV-formatu, sa svim testovima iz označene kampanje i s podatcima
TESTS EXPORT JSON	Generira dokument u JSON-formatu, sa svim testovima iz označene kampanje i s podatcima

Primjer generiranog izvještaja u PDF-formatu prikazan je na slici 4.19. Za potrebe prikaza izgleda izvezenih podataka, kreirana je kampanja s dva testa te su u nastavku prikazani izgledi pojedinih formata.

Dio izvezenih podataka u XLSX-formatu prikazan je na slici 5.7.

CampaignName	AverageDuration	IntendedRunType	BoardName	TesterName	ShiftName	StartTime
PokaznaKampanja1	00:06:00	manual	TS1	Denis Ivanovic	SecondShift	2022-09-02 16:00:00+00:00
PokaznaKampanja1	00:06:00	manual	TS1	Denis Ivanovic	SecondShift	2022-09-02 17:00:00+00:00

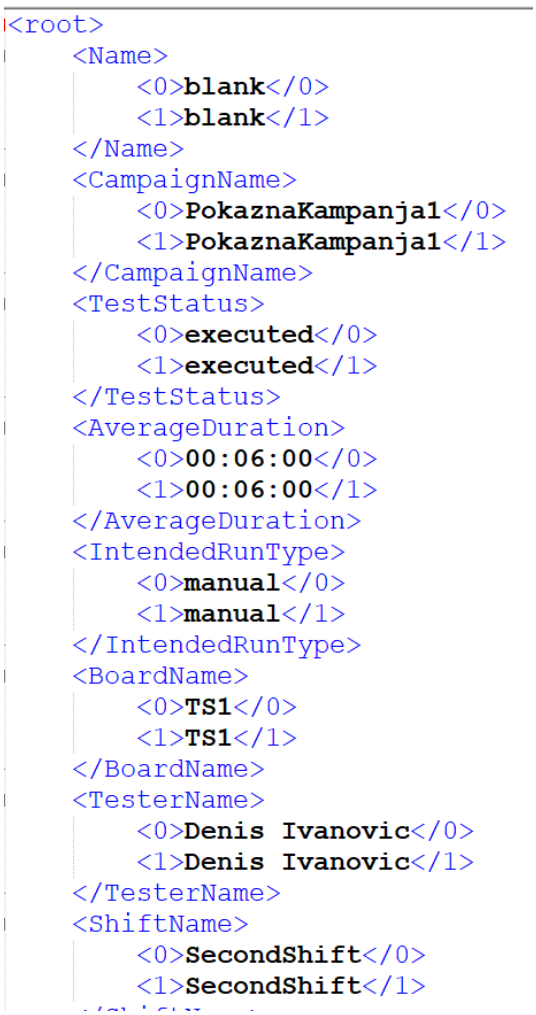
Sl. 5.7. Dio izvezenih podataka o testovima u XLS- formatu.

Dio izvezenih podataka u CSV-formatu prikazan je na slici 5.8.

```
,Name,CampaignName,TestStatus,AverageDuration,IntendedRunType,BoardName,TesterName
0,blank,PokaznaKampanja1,executed,00:06:00,manual,TS1,Denis Ivanovic,SecondShift,a
1,blank,PokaznaKampanja1,executed,00:06:00,manual,TS1,Denis Ivanovic,SecondShift,m
```

Sl. 5.8. Dio izvezenih podataka o testovima u CSV-formatu.

Dio izvezenih podataka u XML-formatu prikazan je na slici 5.9.



Sl. 5.9. Dio izvezenih podataka o testovima u XML-formatu.

Dio izvezenih podataka u JSON-formatu prikazan je na slici 5.10.

```
{
    "Name":{
        "0":"blank",
        "1":"blank"
    },
    "CampaignName":{
        "0":"PokaznaKampanja1",
        "1":"PokaznaKampanja1"
    },
    "TestStatus":{
        "0":"executed",
        "1":"executed"
    },
    "AverageDuration":{
        "0":"00:06:00",
        "1":"00:06:00"
    },
    "IntendedRunType":{
        "0":"manual",
        "1":"manual"
    },
    "BoardName":{
        "0":"TS1",
        "1":"TS1"
    },
    "TesterName":{
        "0":"Denis Ivanovic",
        "1":"Denis Ivanovic"
    },
    "ShiftName":{
        "0":"SecondShift",
        "1":"SecondShift"
    },
```


Sl. 5.10. Dio izvezenih podataka o testovima u JSON-formatu.

6. ZAKLJUČAK

Iako postoji mnoštvo sličnih alata onom napravljenom kroz obradu teme ovog diplomskog rada, može se zaključiti da i dalje ima potrebe za razvijanjem vlastitih alata. Gotovi komercijalni alati su preskupi i nefleksibilni. Napravljen je pregled područja teme ovog rada, razmotreni su i spomenuti neki od najpopularnijih i najsličnijih komercijalnih programskih rješenja. Također je napravljen uvid u razloge razvijanja vlastitog rješenja. Prikazani su početni zahtjevi koje bi sustav trebao zadovoljavati, postupak osmišljanja izgleda sustava i funkcijske povezanosti modula. Dizajnirano je rješenje s tri glavna modula; modul za generiranje testnih kampanja, modul za grafički prikaz rezultata testova i podataka i modul za generiranje izvještaja i izvoz podataka. Glavni moduli razvijeni su prema dizajnu i uz pomoć ostalih, manjih pomoćnih modula, čine softversko rješenje.

Kreirani i izvršeni testovi automatski su spremljeni u bazu podataka, iz kojih ih se i dalje obrađuje, kako je i postavljeno u zahtjevima pristupa. Podatci su prikazani putem izvještaja, vidljivi su direktno putem internetskog sučelja te ih je u raznim oblicima moguće preuzeti na računalo korisnika. Izvještaj generiran u PDF-formatu prikazan je na jednoj stranici A4-formata i sadrži osnovne pokazatelje izvršene kampanje. Da bi se ovaj izvještaj upotpunio, prvo bi bilo potrebno u bazu podataka dodati neke tablice i spremati dodatne podatke, koji bi se onda preračunavali i ubacili u izvještaj. Jedno od navedenog je kreiranje dodatnih dviju tablica u bazi podataka o detaljnim zauzećima testera i testnih stanica, kako bi se za svako zauzeće i izvršenje svakog testa u bazi moglo zapisati vrijeme početka i kraja. Uz te podatke, mogao bi se razviti algoritam koji proračunava i prati kada je tester ili testna stanica slobodna, te koji je odnos učinka planiranog i stvarnog vremena izvršavanja. Jedan od glavnih segmenata ovog programskog rješenja je rad s vremenima, što je nepredvidivo i teško upravljivo, jer se vremena iz baze podataka i iz polja za unos čitaju u obliku teksta. Da bi bilo moguće učiniti izračune i procjene, takve je podatke potrebno neprestano prebacivati u odgovarajući zapis vremena. Prema tome bi još jedan način za unaprjeđenje softvera bio taj da se upravljanje i pretvaranje vremena prebaci u zaseban modul, što bi pojednostavilo korištenje i ubrzalo rad.

Naravno da i u ovom rješenju ima mjesta za unaprjeđenje i refaktoriranje programskog kôda i razvoj boljih algoritama za obradu podataka. Svako unaprjeđenje i nadogradnja mogu samo poboljšati brzinu i točnost softvera, te su oni nužni zbog konstantnog mijenjanja uvjeta i zahtjeva.

LITERATURA

[1] „Table 1-23: World Motor Vehicle Production, Selected Countries (Thousands of vehicles)", Bureau of Transportation Statistics, dostupno na: https://www.bts.gov/bts/archive/publications/national_transportation_statistics/table_01_23 [pristupano 13. rujna 2022.].

[2] B. Fleming, „Microcontroller Units in Automobiles [Automotive Electronics] ", IEEE Veh. Technol. Mag., sv. 6, izd. 3, str. 4–8, ruj. 2011, doi: 10.1109/MVT.2011.941888.

[3] „The Economic Impacts of Inadequate Infrastructure for Software Testing, National institute of standards and technology, dostupno na: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf [pristupano: 31. kolovoza 2022.].

[4] „Introduction to TestRail", Gurock Software GmbH, dostupno na: https://support.gurock.com/hc/en-us/articles/7076810203028-Introduction-to-TestRail [pristupano 30. kolovoza 2022.].

[5] „TestRail: Test Management & QA Software for Agile Teams", Gurock Software GmbH, dostupno na: https://www.gurock.com/testrail/ [pristupljeno 01. rujan 2022.].

[6] „Gurock", Gurock Software GmbH, dostupno na: https://support.gurock.com/hc/en-us [pristupano 30. kolovoza 2022.].

[7] „Full API Documentation (Version 2) - PractiTest", H.S. PractiTest Ltd., dostupno na: https://www.practitest.com/api-v2/ [pristupano 30. kolovoza 2022.].

[8] „PractiTest - QA Test Management Tool that works for You", H.S. PractiTest Ltd., dostupno na: https://www.practitest.com/ [pristupano 1. rujna 2022.].

[9] „Zephyr Enterprise", SmartBear Software, Inc., dostupno na: https://smartbear.com/test-management/zephyr-enterprise/ [pristupano 01. rujna 2022.].

[10] „Zephyr Documentation - Zephyr Documentation", SmartBear Software, Inc., dostupno na: https://zephyrdocs.atlassian.net/wiki/spaces/ALLDOCS/overview?mode=global [pristupano 31. kolovoza 2022.].

[11] „Okta", Kualitatem Inc., dostupno na: https://www.kualitee.com/user-guide/ [pristupljeno 31. kolovoza 2022.].

[12] „Best Test Management Software & Test Case Management Tool", Test Collab Software Inc, dostupno na: https://testcollab.com/ [pristupano 1. rujna 2022.].

[13] „Test Collab API", Test Collab Software Inc, dostupno na: https://api.testcollab.com [pristupano 31. kolovoza 2022.].

SAŽETAK

Primjereno praćenje vremena trajanja testova glavni je preduvjet za poboljšanje njihovim upravljanjem. U ovom radu obrađivana je tema koja se odnosi na praćenje trajanja testova u razvoju programske podrške u automobilskoj industriji. Problem praćenja testova posebno je aktualan u automobilskoj industriji, zbog teško dostupnog hardvera na kojem se testiranje radi. Zato je potrebno dobro organizirati izvršenje velikog broja testova sa, često, malo opreme i manjkom testera u što kraćem vremenskom roku, a kvalitetno. Problem je rješavan tako da se svako izvršenje testa upisivalo u bazu podataka i za svaki se test pratilo vrijeme početka, kraja, tko ga je izvršio i na kojoj opremi. Tako zabilježeni rezultati formatirani su, obrađeni i prikazani kroz izvještaje i grafičke prikaze, te uz pomoć njih možemo uvidjeti gdje ima prostora za poboljšanje.

Ključne riječi: automobilska industrija, grafički prikaz, izvještaj, praćenje vremena, testiranje, upravljanje testovima

ABSTRACT

Title: Monitoring the Duration of Tests in the Development of Program Support in the Automobile Industry

Adequate monitoring of test duration is the main prerequisite for their improvement. In this paper, the topic covered is based on the duration of tests during the development period of program support for the automotive industry. The problem of monitoring tests is especially present in the automotive industry, all because of very difficult access to the hardware on which the tests were performed. For this reason, it is necessary to establish a very good organization to ensure as many tests as possible, with the lack of equipment and testers, in a short period of time. The problem was solved by putting every test that was made into a database, tracking when the test started, when it ended, who did the test and on what equipment. The results recorded after testing are formatted, processed and presented in reports and charts that help show if there is room for improvement.

Key words: Automotive Industry, Graphical Display, Report, Time Tracking, Testing, Test Management

ŽIVOTOPIS

Denis Ivanović rođen je 28. 8. 1998. godine u Požegi. Osnovnu školu Braće Radić pohađao je u Grebnici. Od 2012. do 2016. godine pohađao je srednju školu u Školskom centru fra Martina Nedića u Orašju te stječe zvanje ekonomskog tehničara. Od 2016. do 2019. godine pohađa preddiplomski stručni studij elektrotehnike, smjer Informatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, gdje stječe zvanje prvostupnog inženjera elektrotehnike. U akademskoj godini 2019./2020. pohađa i završava Razlikovne obveze smjer Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. 2020. godine upisuje diplomski sveučilišni studij računarstva smjer Programsko inženjerstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Tijekom pohađanja fakulteta postiže znanja iz područja programskog inženjerstva, koja se odnose najviše na programiranje programskim jezicima *Java,* C i *Python*.

_____

Potpis autora