Sprega HbbTV testnog sustava sa sustavom za automatsko testiranje

Plenković, Marko

Master's thesis / Diplomski rad

2016

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:200:863655

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-01-09**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Sveučilišni studij

Sprega HbbTV testnog sustava sa sustavom za automatsko testiranje

Diplomski rad

Marko Plenković

Osijek, 2016.

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek,

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Marko Plenković
Studij, smjer:	Sveučilišni diplomski studij, Komunikacije i informatika
Mat. br. studenta, godina upisa:	D-864, 2014
Mentor:	Doc.dr.sc. Marijan Herceg
Sumentor:	MSc. Dejan Stefanović
Predsjednik Povjerenstva:	Doc.dr.sc. Mario Vranješ
Član Povjerenstva:	Doc.dr.sc. Ratko Grbić
Naslov diplomskog rada:	Sprega HbbTV testnog sustava sa sustavom za automatsko testiranje
Primarna znanstvena grana rada:	Telekomunikacije i informatika
Sekundarna znanstvena grana (ili polje) rada:	Informacijski sustavi
Zadatak diplomskog rada:	Izraditi spregu sustava za automatsko testiranje s uređajem koji će se testirati i pri tome automatizirati što veći dio testiranja. Također je potrebno izraditi spregu i korisničko sučelje sustava.
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 Postignuti rezultati u odnosu na složenost zadatka: 3 Jasnoća pismenog izražavanja: 3 Razina samostalnosti: II

Potpis su mentora:	Potpis mentora:

Dostaviti:

1. Studentska služba

U Osijeku, 2016 godine

Potpis predsjednika Odbora:

ETFOS
ELEKTROTEHNIČKI FAKULTET OSIJEK
Sveučilište Josipa Jurja Strossmayera u Osijeku

IZJAVA O ORIGINALNOSTI RADA

Osijek,

Ime i prezime studenta:	Marko Plenković
Studij :	Diplomski studij, Komunikacije i informatika
Mat. br. studenta, godina upisa:	D-864

Ovom izjavom izjavljujem da je rad pod nazivom**: Sprega HbbTV testnog sustava sa sustavom za automatsko testiranje**

izrađen pod vodstvom mentora: doc.dr.sc. Marijana Hercega

i sumentora MSc. Dejana Stefanovića

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. Uvod ... 1

2. Teorijske osnove ... 2

2.1 Black box testing ... 2

2.2 HbbTV ... 4

2.3 Sustav za testiranje (RT-Harness) ... 5

2.4 Arhitektura HbbTV sustava ... 8

2.5 Korisničko sučelje ... 11

3. Izvedba sprege sustava ... 16

3.1 Programsko okruženje ... 16

3.2 Polymer biblioteka ... 17

3.3 Node.js server i SQLite baza podataka ... 20

4. Koncept rješenja ... 25

4.1 Opis modula aplikacije ... 25

4.1.1 Polymer ... 25

4.1.2 Node.js poslužitelj ... 27

4.1.3 TestAPI datoteka ... 28

4.2 Testiranje aplikacije ... 29

5. Zaključak ... 32

Literatura ... 33

Sažetak ... 34

Summary ... 35

Životopis ... 36

1. Uvod

Za potrebe certifikacije DTV uređaja, HbbTV (engl. Hybrid Broadcast Broadband TV) udruženje definira hardversko/softversko okruženje u kojem se izvršava certificiranje, kako bi proizvođači uređaja imali mogućnost testirati svoje uređaje tijekom razvoja u prikladnom okruženju. Spomenuto okruženje ne uključuje implementaciju, tako da proizvođači DTV uređaja mogu koristiti neku od postojećih implementacija okruženja ili implementirati svoje okruženje u kojem se dani testovi mogu izvršavati automatski ili ručno. U radu je napravljen pregled HbbTV sustava za testiranje i analiza mogućnosti automatizacije u odnosu na postojeći test API. Potom je proširen postojeći sustav za HbbTV testiranje s mogućnosti kreiranja testnih planova i integrirana je razmjena i izvršenje testnih planova s postojećim sustavom za automatsko testiranje DTV uređaja u skladu s rezultatima prethodne analize.

Naglim razvojem tehnologije se postavljaju sve veći zahtjevi pred pružatelje TV usluga. Traže se Smart funkcionalnosti, programski vodič, mogućnost pretraživanja sadržaja i slično što postavlja velike hardverske zahtjeve. Zbog sve većih zahtjeva nastao je HbbTV koji objedinjuje funkcionalnosti jednostavnog televizijskog prijemnika sa proširenim Smart funkcijama. HbbTV nudi puno više od standardnog prijemnika, te omogućava interaktivnost korisnika sa sadržajem što je prije toga bilo nemoguće, jer nije postojala povratna veza od korisnika prema davatelju usluge. Uvođenjem HbbTV standarda se ova barijera miče i korisnik može sam birati sadržaj koji želi gledati (*Pay Per View - PPV*), postoji virtualna videoteka iz koje korisnik može odabrati sadržaj koji ga zanima i koji želi gledati, uvedene su aplikacije koje omogućavaju prikaz vremenske prognoze, stanja na cestama i mnoge druge korisne informacije. Kako bi se omogućila podrška za sve navedene aplikacije proizvođači su definirali programsku podršku zasnovanu na Linux operativnom sustavu, te se dodatno koristi Android4TV kao programsko rješenje.

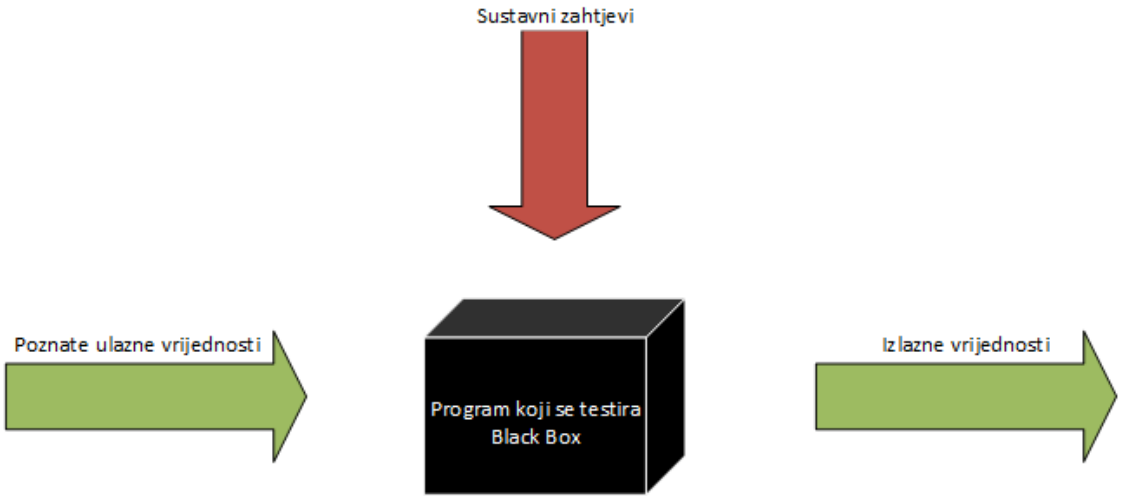
2. Teorijske osnove

U ovom poglavlju će biti napravljen kratak opis razvoja digitalne televizije i interneta, osnove HbbTV standarda, principi i pristup *Black Box Testiranju,* te postupak izvođenja testova. Bit će navedene osnove HTTP protokola na kojemu se zasniva trenutno postojeće rješenje za izvođenje testova.

2.1 Black box testing

Testiranje programa je postupak provjere ispunjava li program sve što je navedeno u sistemskim zahtjevima. Verifikacija i validacija programa su ključni koraci za ostvarivanje visoke kvalitete usluge. Dvije glavne filozofije testiranja su crna kutija (eng. *black box, BBT*) i bijela ili prozirna kutija (eng. *white box, WBT*). Kod BBT testiranja, osoba koja testira dio koda ili programa ne mora poznavati unutarnju strukturu koda, te u krajnjem slučaju ne bi ni trebala imati pristup unutarnjim funkcionalnostima programa kako bi se zaštitilo intelektualno vlasništvo i spriječila mogućnost krađe te se može izvršiti testiranje neovisno o razini znanja, dok kod WBT je potrebno da osoba koja testira program poznaje kako program funkcionira i ima potrebno znanje za uočavanje pogrešaka u samom kodu te osobe koje testiraju dio koda su najčešće programeri koji su sami pisali dio koda.

BBT je sinonim za filozofiju testiranja uređaja ili *softwera* korištenjem poznatih ulaznih vrijednosti i promatranja vrijednosti i pojava na izlazu iz sustava. Tester ne mora poznavati unutarnju strukturu ili kod koji obavlja neku funkciju unutar samog programa koja obavlja ono što je potrebno za funkcioniranje programa. Zbog toga što je unutarnja struktura i kod nepoznat osobi koja testira uređaj ova testna filozofija se naziva BBT. Ova filozofija testiranja se koristi kod viših stadija razvoja programa, dok se u početnim fazama koristi *white box*.



Slika 2.1. Vizualna predodžba BBT-a

Ova vrsta testiranja se naziva i funkcionalno testiranje jer se bazira na provjeri radi li program ono što je definirano specifikacijom, te pokušava pronaći pogreške u vanjskom ponašanju programa tj. rezultatima koje program daje nakon neke poznate i unaprijed definirane pobude. Tokom testiranja se traže pogreške u sljedećim kategorijama:

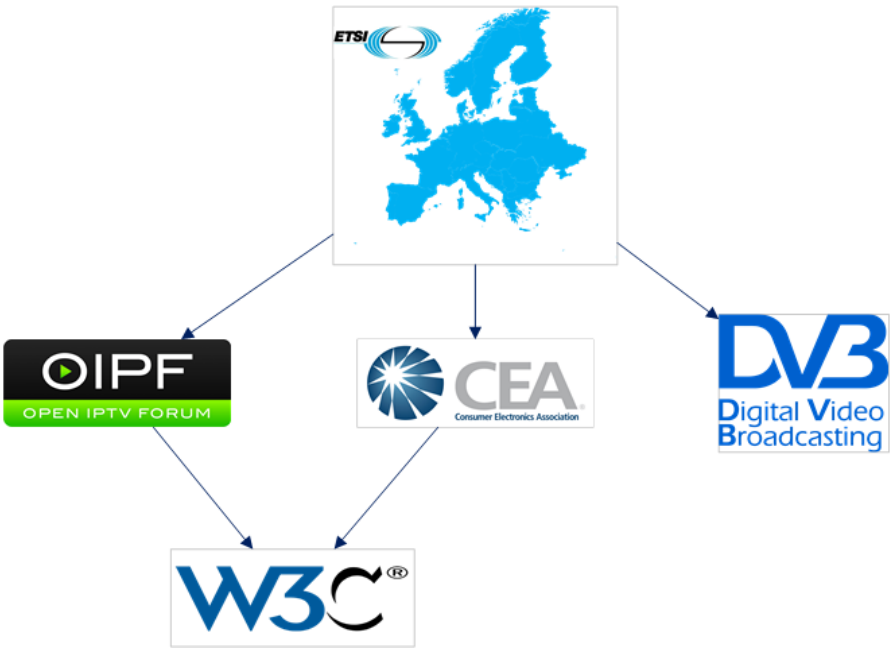
- netočna ili nepostojeća funkcionalnost
- pogreške unutar sučelja
- pogreške u podatcima korištenim od strane sučelja
- pogreške u ponašanju ili performansama
- pogreške pri pokretanju ili gašenju

Uzimajući u obzir sve rezultate testiranja može se pretpostaviti da program funkcionira kako je zamišljeno i definirano, ali bez obzira na količinu testiranja, nije moguće sa sigurnošću tvrditi da program nema grešaka. Testovi koji daju konačnu prolaznu ocjenu su napisani od strane korisnika, što znači da korisnik može odbiti isporuku programa u slučaju da program ne radi kako je definirano ili ne prolazi testove koji su napisani za provjeru programa.

Automatiziranje testova može biti izazovan posao jer osoba koja piše testove ne poznaje unutarnju strukturu i ponašanje nekog uređaja ili dijela programa koji se testira, ali što je bolja automatizacija testova, lakše je pokretanje testnih slučajeva i uzastopno pokretanje kako bi se mogla dobiti potvrda da će program na pojedinu pobudu svaki puta davati jednak rezultat na izlazu.

2.2 HbbTV

HbbTV je standard i inicijativa za razvoj i usvajanje hibridne televizije kako bi se stopila granica između klasičnog odašiljanja, IPTV i širokopojasnih usluga sa željom pružanja što bolje usluge krajnjem korisniku koji ima priključen *Smart* TV ili *set-top box* koji posjeduje hardverske i softverske sposobnosti da iskoristi navedene usluge. HbbTV je nastao kao suradnja i spajanje dva projekta (H4TV i HTML profil), te se tokom 2014. godine došlo do spajanja s Open IPTV Forumom, sličnoj organizaciji koja je radila na prijenosu televizije putem Interneta [2].



Slika 2.2. Standardi i udruženja pod okriljem HbbTV-a

HbbTV omogućuje korištenje usluga koje nisu podržane putem standardnog odašiljanja, te samim time poboljšava korisnikovo iskustvo. Neke od tih usluga su :

- napredni teletekst
- različite usluge za snimanje video sadržaja
- video na zahtjev

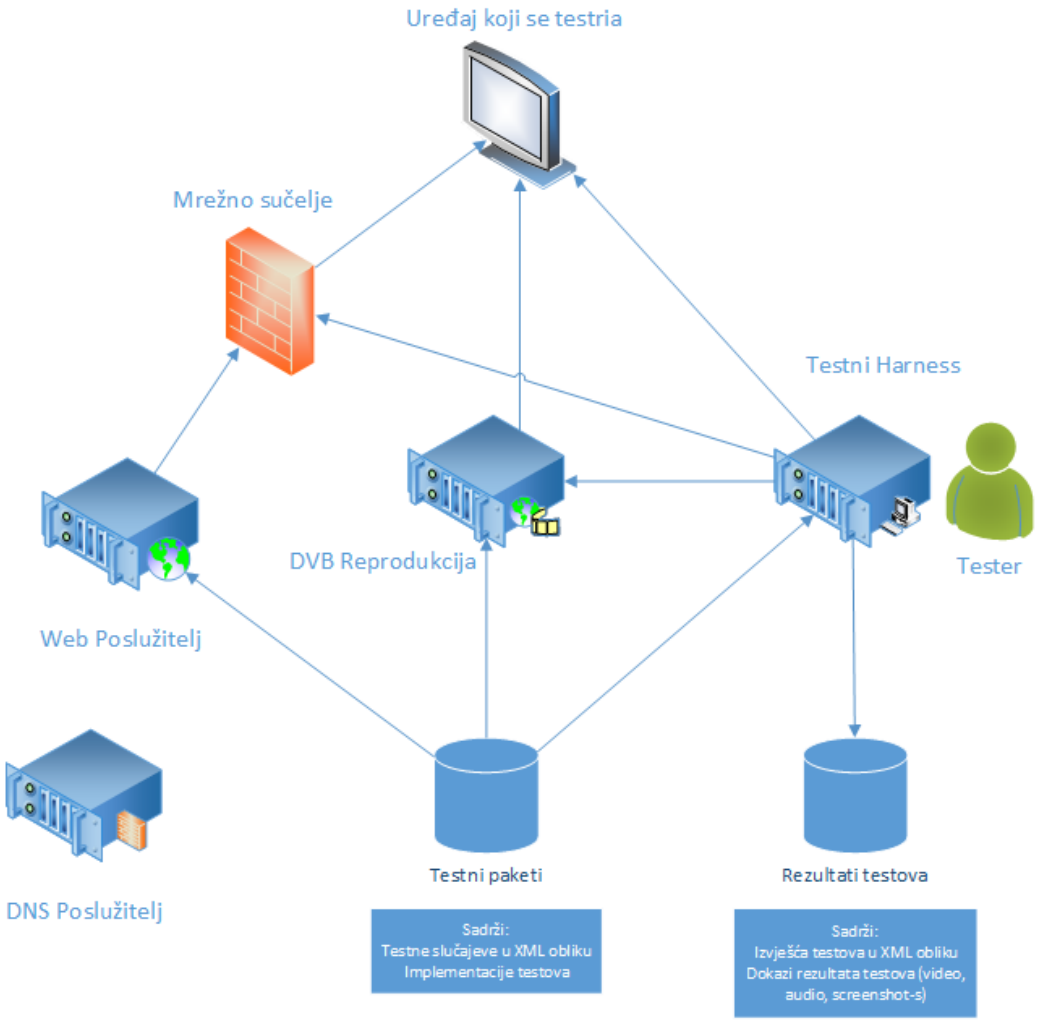
- elektronski programski vodič
- interaktivno oglašavanje
- personalizacija
- glasanje
- igre
- društvene mreže

Prednosti hibridne televizije su dostupnost velike količine naprednog interaktivnog sadržaja i raznih korisnih servisa i pristup informacijama, sve putem samo jednog uređaja. HbbTV konzorciji obuhvaća više od 50 tijela koja podupiru razvoj standarda uključujući standardizacijska tijela, TV kuće, davatelje usluga uvjetnog pristupa, proizvođače uređaja uvjetnog pristupa, testna udruženja te mnoge druge. Zbog tolike raznolikosti podupirućih tijela, HbbTV standard se rapidno razvija i postiže velik uspjeh i podršku globalne zajednice. Velik dio standarda je otvoren i omogućava razvijanje aplikacija i podrške za HbbTV od strane korisnika i nezavisnih programera. MHEG je britanski standard za hibridnu televiziju koji nije postigao jako velik uspjeh zbog činjenice da je standard zatvoren i karakterizira ga svojstveni programski jezik za pisanje aplikacija. Samim time je vrlo nepristupačan i sužava količinu slobodnih programera koji imaju potrebna znanja da programiraju aplikacije i podršku za taj standard.

2.3 Sustav za testiranje (RT-Harness)

Sustav za testiranje je paket alata koji omogućuju suglasnost specifikacija proizvođača sa specifikacijama HbbTV standarda, te omogućuje provođenje specifičnih scenarija kako bi se utvrdila ispravnost uređaja. Unutar alata je moguće kreiranje testnih planova kako bi se omogućilo automatsko testiranje većeg broja testnih slučajeva, u svrhu smanjivanja opterećenja na osoblje koje testira uređaje. Testni plan može sadržavati nekoliko desetaka ili stotina testnih slučajeva koje bi tester u protivnom morao pojedinačno pokretati. Testiranje uređaja referencira nekoliko postojećih standarda iz W3C konzorcija koji standardizira korištenje elemenata na web-u. HbbTV aplikacije su tehnički vrlo slične ostalim web aplikacijama, ali je korisničko iskustvo vrlo različito. Aplikacija se tipično pokreće direktno iz širokopojasnog kanala gdje korisnik nema percepciju da koristi nešto poput mobilne ili PC aplikacije. Zbog ovoga je ključno da

aplikacije rade bez grešaka i zahtjeva za korisničkom akcijom. Velik broj različitih proizvođača i opreme koja koristi navedeni standard, potrebno je testirati ispravnost aplikacije na različitim uređajima i njenu suglasnost sa standardom. Zahtjevi za testiranjem proizlaze iz potrebe da davatelji usluga, nacionalna tijela za regulaciju i proizvođači opreme mogu garantirati suglasnost svoje opreme i arhitekture sa standardnom. Ovo se postiže izvršavanjem potrebnog broja testova na uređaju i dobivanjem binarne ocjene (zadovoljio/ne zadovoljava) u konačnom izvješću za testirani uređaj. U nekim slučajevima ocjena može biti preduvjet za pokretanje proizvodnog procesa testiranog uređaja. Proizvođači su osjetljivi na preciznost definiranja koje je testove potrebno izvršiti, te ako se pokaže da je test netočan može brzo doći do povlačenja ili odustajanja od standarda koji se želi postići [1].



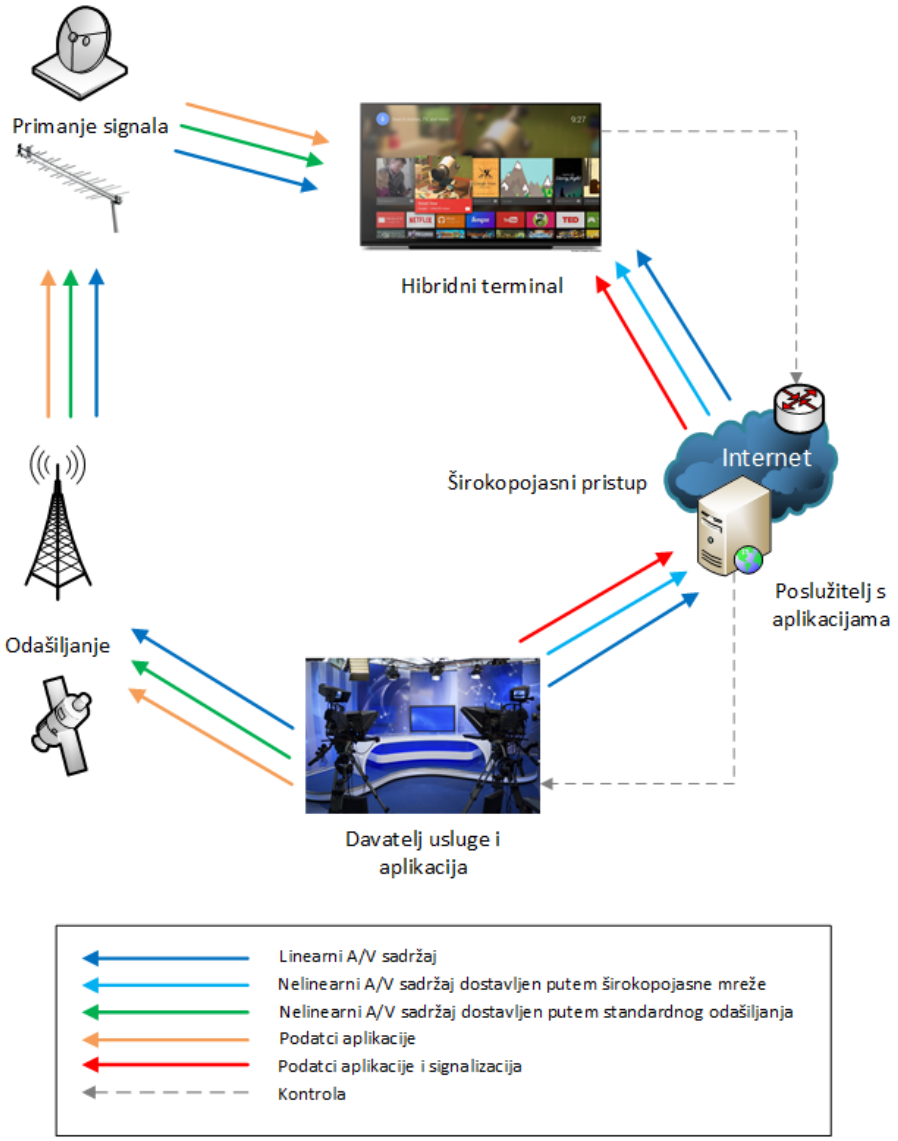
Slika 2.3. Dijagram testnog okruženja

Na slici 2.3 je dan grafički prikaz testnog okruženja koje se koristi za testiranje HbbTV uređaja i davanje prolazne ocjene. Unutar testnog okruženja se nalaze sve komponente potrebne za testiranje uređaja te je moguće dodavanje i adaptacija postojećeg testnog okruženja za buduće standarde. Testni uređaj je povezan na RT-Harness koji kontrolira osoba koja testira uređaj, web poslužitelj i na izvor DVB signala. Sve komponente su međusobno povezane bazom podataka iz koje dohvaćaju testne pakete koji su unaprijed definirani za provođenje specifičnih testova. Unutar baze podataka su sadržani podatci o testovima, testni slučajevi, njihove implementacije koje omogućuju pokretanje testova unutar Harness-a, te izvješća o testovima, rezultate testova u XML obliku, te dokaze o rezultatima testova. Svaki test mora zadovoljavati unaprijed definirane stavke postavljene od HbbTV konzorcija kako bi testiranje bilo uniformno za sve proizvođače i uređaje. Neki od tih zahtjeva su :

- Svaki test mora sadržavati jedinstveni ID koji ga razlikuje od ostalih testova
- Objašnjenje testa kako bi tester znao što se točno testira
- Preduvjete za uspješno provođenje testa
- Test se mora brzo izvršiti, koristeći automatizaciju gdje god je moguće
- Test se mora izvršavati na različitim testnim sustavima
- Popis testnih slučajeva mora biti precizno definiran i verzioniran u bilo kojem trenutku kako bi svi uređaji bili testirani na istoj osnovi
- Moguće je pokrenuti testove selektivno u svrhu provjere zadovoljavanja specifičnog dijela standarda
- Testni sustav se može pokretati na lokalnom mrežnom sučelju kako bi se testiranje izoliralo od nepredvidivog mrežnog ponašanja
- Pokretanje testova na uređaju ne zahtjeva nikakav poseban program na uređaju koji se testira
- Testni sustav pruža potpuno, računalno čitljivo izvješće koje sadrži rezultate o suglasnosti sa standardom, te mora pružiti rezultate da su se točni testovi pokrenuli na uređaju, zajedno sa detaljnim ispisom svih rezultata

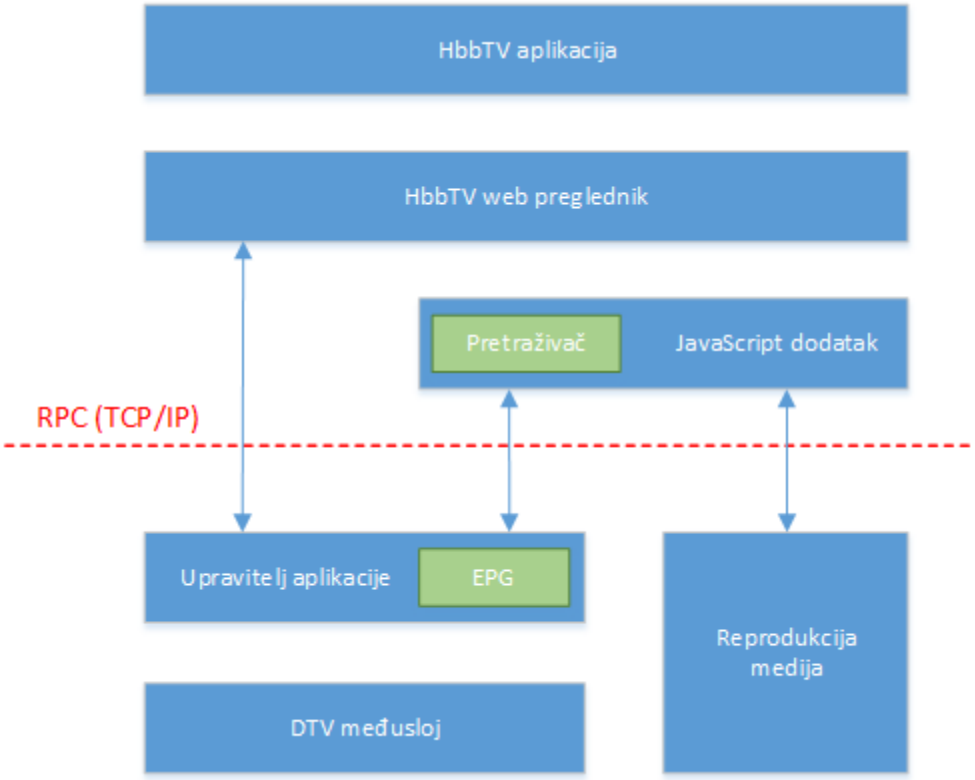
2.4 Arhitektura HbbTV sustava

Zbog koncepta HbbTV sustava da uređaj koristi Internet i prima televizijske signale za ispravan rad aplikacije, potrebno je osigurati kompletnu mrežnu infrastrukturu za pravilno funkcioniranje sustava. Uređaji koji koriste Internet i širokopojasnu mrežu se nazivaju hibridni terminali. Rad terminala je moguć u slučajevima kada jedno od mrežnih sučelja nije dostupno, ali će funkcionalnosti uređaja biti ograničene. Uređaj se spaja na neku od DVB mreža (DVB-T, DVB-T2, DVB-C, DVB-S i sl.) i na Internet. Na taj način je omogućena dvosmjerna komunikacija između terminala i davatelja usluga [2].



Slika 2.4. Prikaz infrastrukture HbbTV sustava

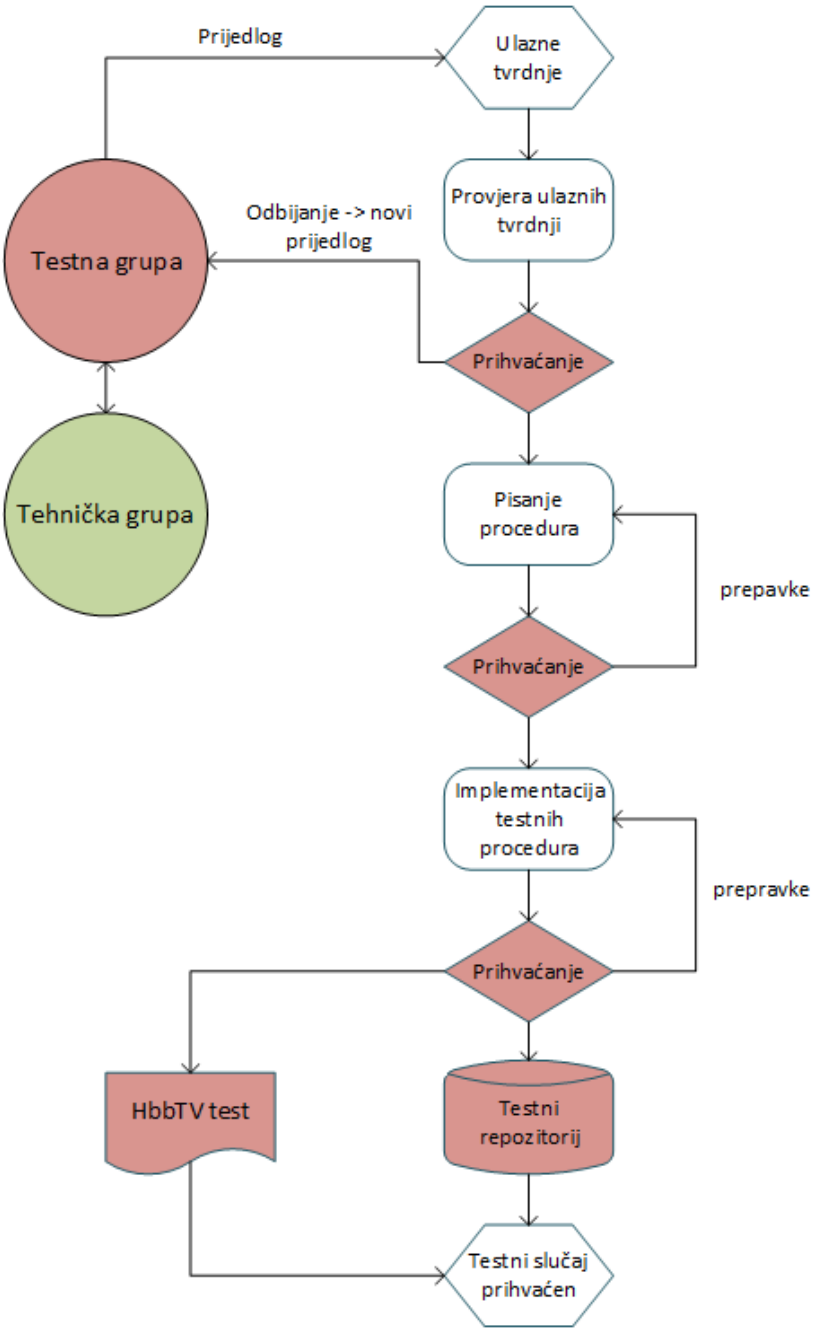
U slučaju kada je hibridni terminal spojen na Internet, omogućene su funkcionalnosti koje nemaju stvarno-vremensku komponentu sadržaja. Moguće je primanje zvučnih ili video zapisa na zahtjev korisnika. Ovo širokopojasno sučelje omoguće spajanje partnerskog zaslona, te je moguće sinkronizirati sadržaj hibridnog terminala na uređaj poput tableta ili mobitela za praćenje na više uređaja. Sve HbbTV aplikacije namijenjene radu unutar web preglednika te su pisane korištenjem JavaScripta. Ovo omogućuje jednostavniju integraciju aplikacija, jer su aplikacije ustvari proširenja za web preglednik te se bitno ne razlikuju od proširenja za preglednike na osobnim računalima. Na slici 2.5 je prikaz programskog stoga koji se koristi u ostvarivanju funkcionalnosti hibridnog terminala.



Slika 2.5 Arhitektura HbbTV sustava

Kako bi svi testovi za ispitivanje ispravnosti web preglednika, pretraživača i ostalih dodataka bili u skladu sa standardnom, u specifikaciji je navedena procedura kreiranja svakog testa. Prijedlog za test dolazi od suradnje tehničke i testne grupi koje rade recenziju predloženog plana koji se nakon toga usvaja ili odbacuje. Ukoliko je test usvojen pišu se testne procedure

nakon kojih slijedi revizija testnih procedura u svrhu utvrđivanja ispravnosti i suglasnosti s željenim standardom. Kada su sve revizije pozitivne, testni slučaj se usvaja i sprema se u testni repozitoriji zajedno s ostalim testovima koji su prošli provjere. Grafički prikaz stvaranje testnog slučaja je prikazan slici 2.6. Testne procedure kreiraju tvrtke koje su dio HbbTV grupacije, dok se recenzija procedura obavlja zajednički.



Slika 2.6. Grafički prikaz stvara testnog slučaja

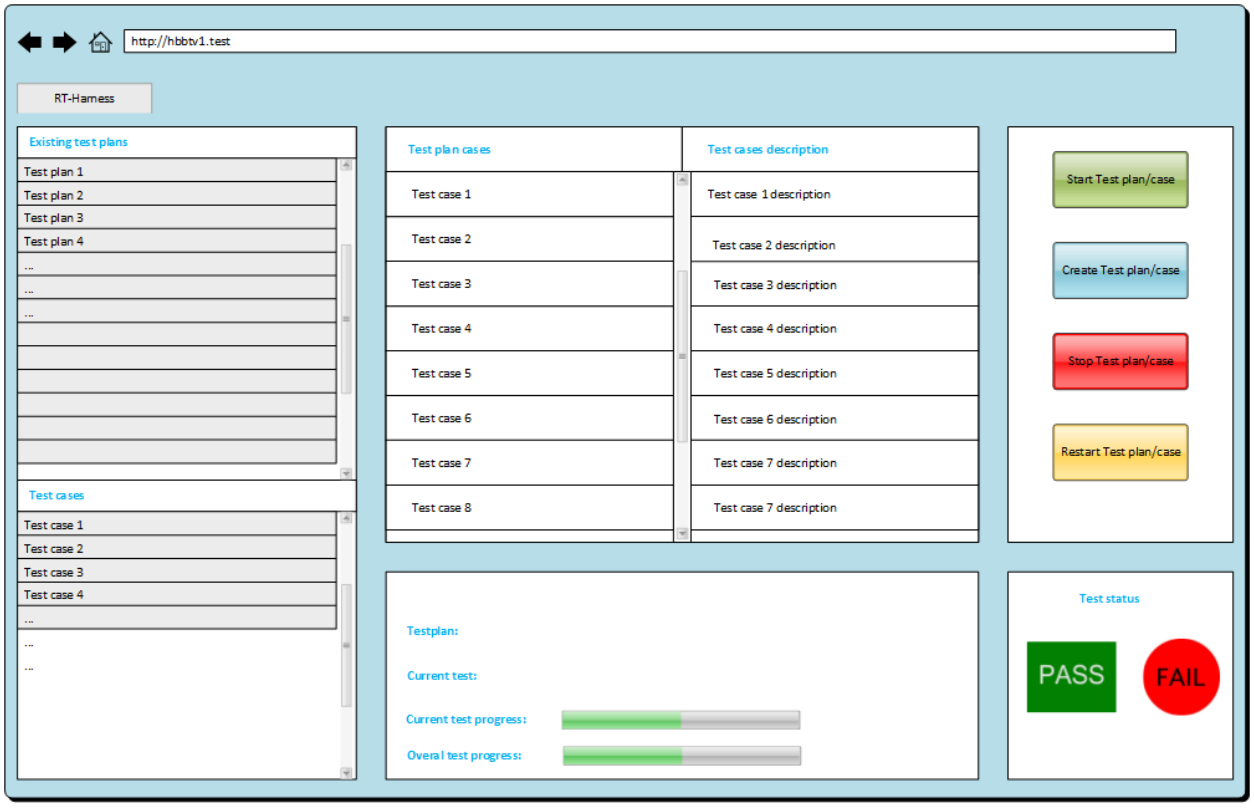
2.5 Korisničko sučelje

Korisničko sučelje je područje gdje uređaj ili računalo i čovjek ostvaruju međusobnu interakciju. Cilj ove interakcije je omogućiti efektivno izvršavanje i kontrolu uređaja od strane čovjeka. Kontrola uređaja zahtjeva povratnu informaciju od strane uređaja kako bi osoba koja upravlja uređajem znala u kojem je stanju uređaj ili program koji se koristi. Najpoznatiji slučaj korisničkog sučelja su vjerojatno operativni sustavi koje u današnje vrijeme gotovo svi koriste. Cilj dobrog dizajna sučelja je stvoriti sučelje koje interakciju s računalom ili uređajem, čini što lakšom, efikasnijom i ugodnijom za osobu koja ga koristi. Postoji 8 glavnih karakteristika koje svako kvalitetno korisničko sučelje treba sadržavati:

- **Jasnoća** – izbjegavanje kompliciranja korištenjem jednostavnog jezika, slijeda rada i hijerarhije
- **Konciznost** – izbjegavanje označavanja svake stvari u sučelju, jer nepoštivanje ovog pravila može dovesti do napučenog sučelja
- **Familijarnost** – korisnik koji prvi puta vidi sučelje treba imati predodžbu o radu sučelja i koje akcije uzrokuju kakve reakcije
- **Odaziv** – kvalitetno sučelje ne smije biti sporo i korisnik ne bih trebao čekati predugo na povratnu informaciju
- **Konzistentnost** – sučelje treba biti ujednačeno kroz cijelu aplikaciju, što omogućuje korisniku da prepozna uzorke
- **Vizualni dojam** – da bi korištenje aplikacije bilo ugodnije korisnicima, potrebno je aplikaciju učiniti vizualno privlačnom
- **Efikasnost** – dobro definirano sučelje treba biti efikasno i korisnika učiniti produktivnijim kroz korištenje
- **Opraštanje** – sučelje ne treba kažnjavati korisnike za njihove pogreške, nego ponuditi načine kako da se iste isprave

Zbog količine uređaja koji će se testirati, potrebno je korisničko sučelje koje je intuitivno, jednostavno i razumljivo, ali se ne smije zanemariti kompleksnost testiranja i stvaranja izvještaja o sukladnosti sa standardom. Sučelje je dio testiranja koje je u direktnoj interakciji s korisnikom, te treba biti ugodno za rad u slučaju dužih testiranja. Pri izradi modela korisničkog sučelja je korišten programski alat Microsoft Visio 2013. Nakon što je model sučelja izrađen i odobren,

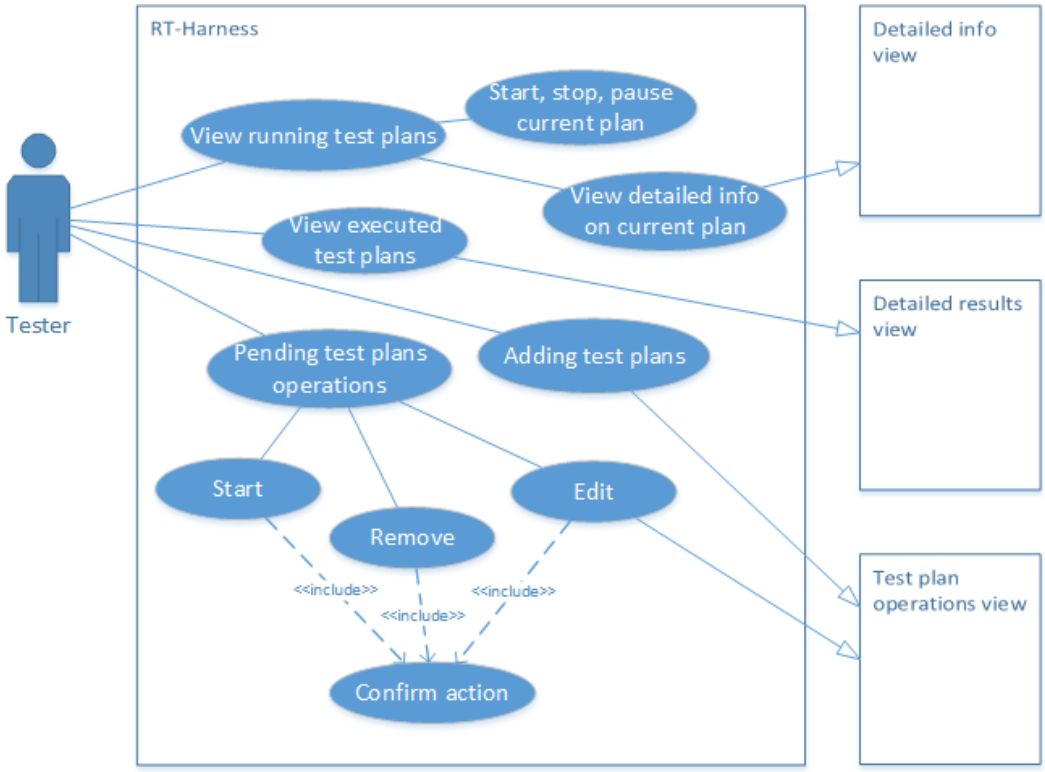
isto sučelje je ostvareno korištenjem programske biblioteke *Polymer*. Sučelje je prošlo kroz 6 većih i više od 10 manjih revizija. Model prvog korisničkog sučelja je prikazan na slici 2.7.



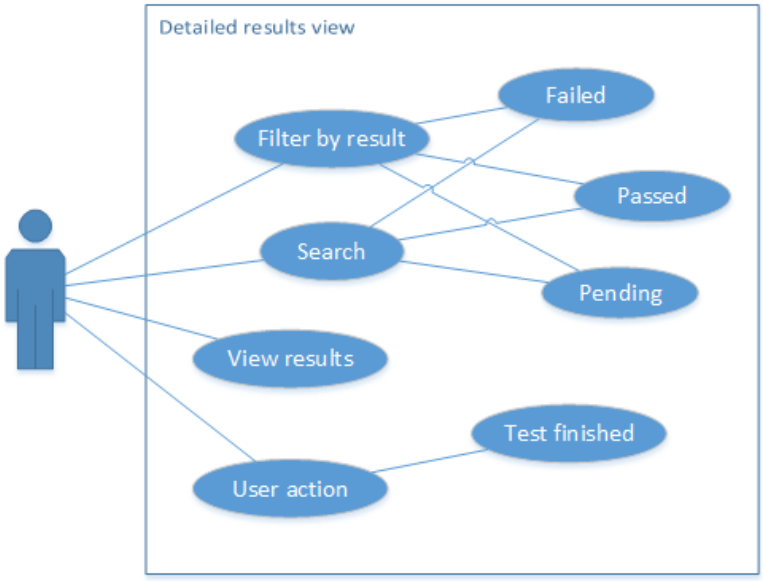
Slika 2.7. Prvi model korisničkog sučelja

Na slici je vidljiva prva iteracija sučelja koja je bila napravljena s idejom jednostavnosti, ali je nepoznavanje funkcioniranja testnog okruženja bio velik nedostatak u dizajniranju, jer ne postoji nikakav prikaz detaljnog izvještaja o testu koji se izvršio nego samo konačan ishod izvedenog testa u binarnom obliku (prošao/pao). Ovakvo sučelje ne zadovoljava sve potrebne informacije koje korisnik treba dobiti u obliku povratnih informacija kako bih imao uvid u ponašanje uređaja. Sljedeće dvije iteracije su bile temeljene na prvom dizajnu i u tome je ključni nedostatak tih iteracija. Prije kreiranja četvrte iteracije sučelja su definirani dijagrami korištenja (eng. *use case diagram*) sa svim mogućim scenarijima koje korisnik može ostvariti. Nakon kreiranja dijagrama je izrađena novi model korisničkog sučelja, temeljen na navedenim dijagramima. Ovakav postupak osigurava sukladnost sučelja sa svim potrebnim zahtjevima, te se

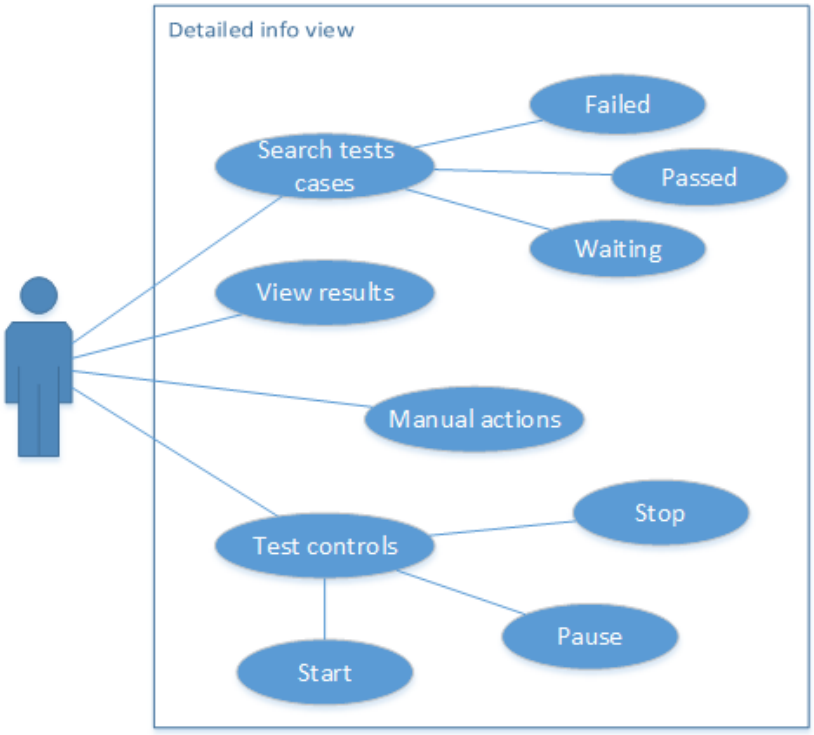
mogu raditi revizije prije nego se izradi sam model, što uvelike olakšava proces modeliranja. Na slikama 2.8 – 2.11 su prikazani dijagrami korištenja prema kojima je modelirano sučelje.



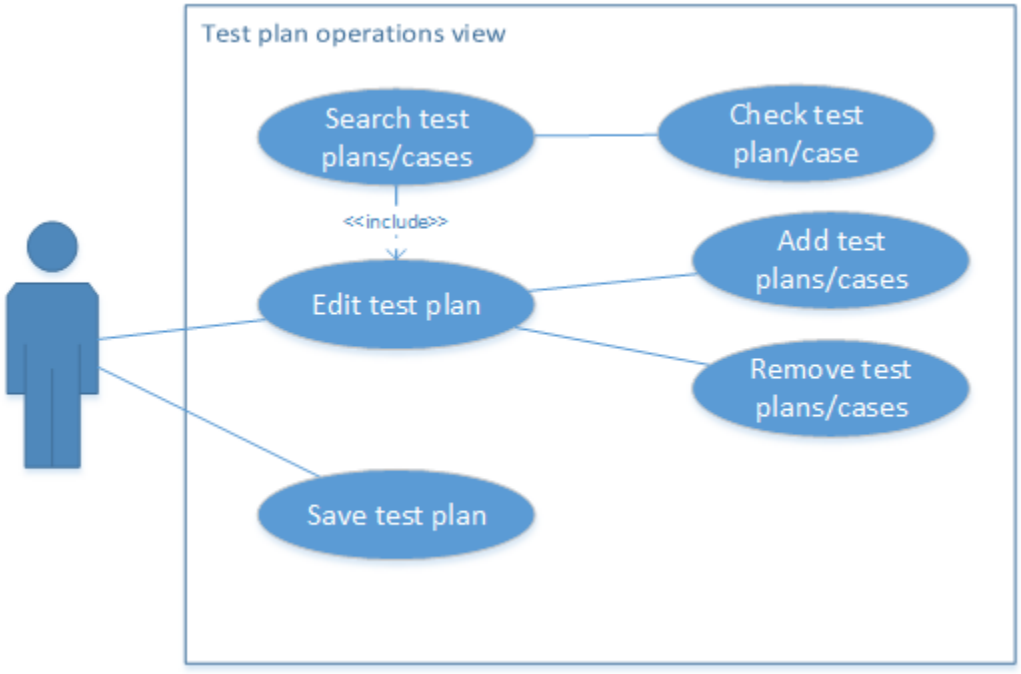
Slika 2.8 Dijagram korištenja za inicijalni prikaz testnog okruženja



Slika 2.9. Dijagram korištenja za detaljni pregled rezultata

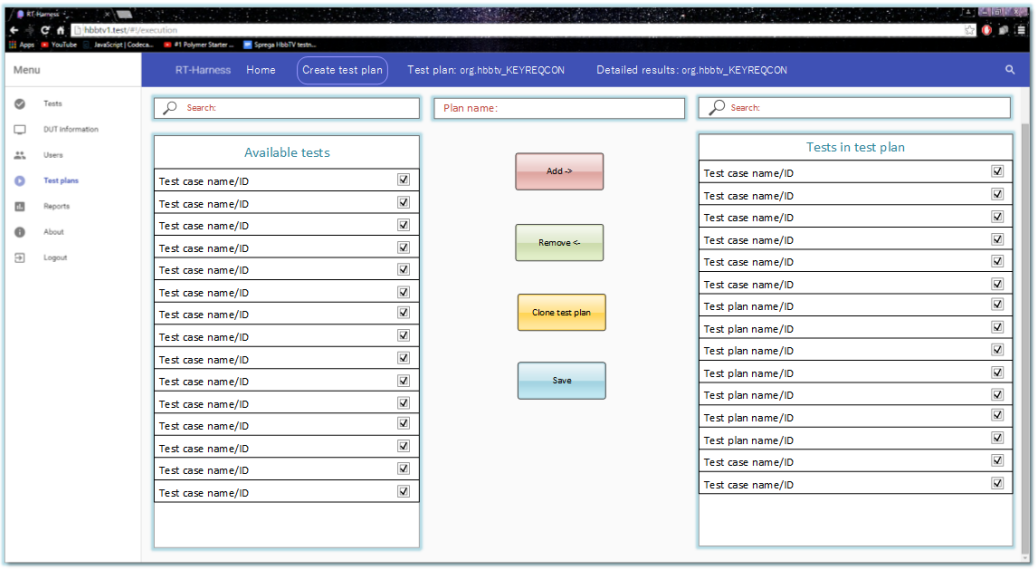


Slika 2.10. Dijagram korištenja za detaljan pregled i kontrolu testova



Slika 2.11. Dijagram korištenja za kreiranje testnih planova

Dijagram na slici 2.8 prikazuje mogućnosti koje su korisniku dostupne pri prvom pokretanju aplikacije. Moguć je pregled postojećih testnih planova, pregled testnih planova koji se izvršavaju i planovi koji čekaju na izvršavanje. Svaka od navedenih akcija ima dalje akcije koje se vežu na njih i omogućuju detaljniji pregled ponašanja aplikacije i izvođenja testova. Ukoliko korisnik odluči za neku od akcija koje izlaze iz glavnog pogleda na aplikaciju, to je omogućeno uz pamćenje položaja i statusa na svakom pogledu koji je korisnik koristio. Kod detaljnog pregledavanja moguće je filtrirati rezultate testnih slučajeva po nekom od stanja u kojemu test može biti (*failed/passed/pending*). U slučaju da test zahtjeva korisničku akciju, kao što je subjektivno uspoređivanje kvalitete slike to je omogućeno korištenjem automatskog procesa koji korisnika navodi na što treba obratiti pozornost. Na taj način osoba koja nema visoku razinu tehničkog znanja može upravljati testiranjem uređaja bez posebne obuke i velikih predznanja. Detaljni pregled rezultata testnog plana koji se trenutno izvodi je također omogućen, te se direktno može upravljati sa testom koji se izvodi, u slučaju kada je to potrebno. Pogled za kreiranje testnih planova se čini najjednostavnijim, ali je posebna pažnja posvećena ovom dijelu sustava za automatsko testiranje zbog imperativa na automatskom testiranju. Broj testnih slučajeva lako može preći 100 jedinica te je potrebno na što lakši i intuitivniji način omogućiti kreiranje testnih planova. Konačni model korisničkog sučelja zauzima previše prostora za čitko prikazivanje zbog svoje veličine i velikog broja opisa pojedinih akcija i mogućnosti. Zbog toga je stavljen kao prilog. Jedan od pogleda sučelja je prikazan na slici 2.12.

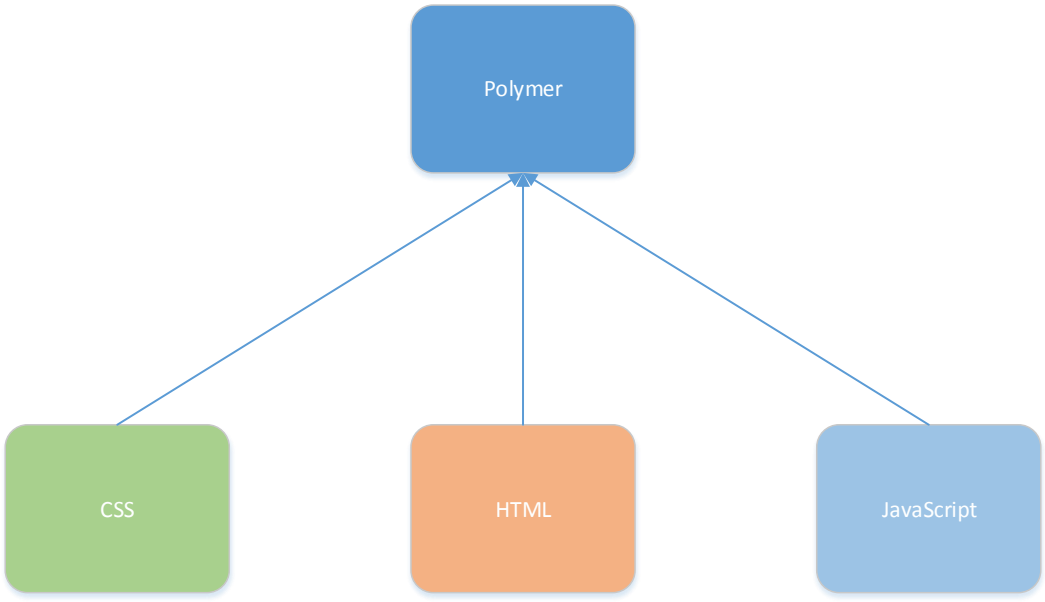


Slika 2.12 Pogled za kreiranje testnih planova

3. Izvedba sprege sustava

3.1 Programsko okruženje

Web aplikacija je izrađena u razvojnom okruženju WebStorm na Windows operativnom sustavu. Serverska strana aplikacije je simulirana korištenjem Node.js platforme i korištenjem lokalnog JSON poslužitelja. Aplikacija se oslanja na korištenje Polymer biblioteke koju je razvio Google. Biblioteka proširuje postojeći HTML i omogućuje korištenje elemenata koje je korisnik sam razvio, te olakšava dodavanje funkcionalnosti na elemente aplikacije. Funkcionalnost aplikacije je ostvarena korištenjem JavaScript skriptnog jezika. Na slici 3.1 je vizualna reprezentacija Polymer biblioteka koja obuhvaća tri velike web tehnologije u jednu i omogućuje stvaranje korisnički definiranih elemenata. Polymer služi prvenstveno za izradu korisničkog sučelja i ostvarivanje funkcionalnosti koje su u direktnoj interakciji s korisnikom. Pri razvoju testne sprege je korišten velik broj biblioteka i razvojnih okruženja od kojih su za rad aplikacije najvažnije *Polymer*, *node.js*, *sequelize*, *express* i još mnogo drugih koje su korištene za vrijeme razvoja u fazama testiranja i rješavanja raznih problema.



Slika 3.1. Blok dijagram Polymera

3.2 Polymer biblioteka

Biblioteka korištena za kreiranje kompletnog korisničkog sučelja je Polymer. Ova biblioteka je relativno nova, te još uvijek dosta često izlaze dodatci i izmjene biblioteke što u nekim slučajevima stvara probleme zbog nekompatibilnosti verzija i elemenata. Polymer je besplatna biblioteka otvorenog koda (eng. *open source*) temeljena na JavaScript-u. Biblioteka omogućuju velik broj funkcionalnosti i uvelike smanjuje vrijeme potrebno za dizajniranje i implementaciju nekog korisnički definiranog elementa. Stvaranje korisničkih elemenata je vrlo jednostavno korištenjem `dom-module` elementa koji Polymer osigurava, kao i dvosmjerno vezanje podataka i izračunavanje svojstava objekta ili elementa. Ukoliko je potrebno koristiti dio koda repetitivno, nema potrebe za gomilanjem koda nego se mogu koristiti predlošci koji se dodaju u slučaju potrebe. Velika prednost pri korištenju ove biblioteke je količina dostupnih elemenata koji su uključeni u Polymer, te se izrada korisničkog sučelja može kretati puno brže nego u slučaju da korisnik mora sam definirati svaki element.

```
<dom-module id="hello-element">
  <template>
    <style>
      /* Local DOM CSS style */
    </style>
    <!-- Local DOM -->
    Hello {{name}}!
  </template>
  <script>
  Polymer({
    is: 'hello-element',
    properties: {
      name: String
      /* Element properties */
    },
    ready: function() {
      /* Called when local DOM is initialized */
    },
    /* Custom methods */
  });
  </script>
</dom-module>
```

Prilog 3.1 Primjer „*Hello world*“ elementa u Polymer-u

U prilogu 3.1 je prikazan jednostavan primjer kreiranja korisničkog elementa i objedinjavanje CSS-a, HTML-a i JavaScript-a unutar jednog elementa. Svaki element se mora definirati korištenjem dom-module elementa s svojstvom imena. Bitno je napomenuti da svaki korisnički element mora imati crticu u imenu kako bi se spriječili mogući konflikti sa već postojećim elementima. Nakon definiranja imena elementa postavlja se `<template>` oznaka unutar koje se piše CSS i HTML kod koji definiraju izgled elementa. U slučaju da korisnik ne želi mijenjati izgled elemenata koje koristi, nije potrebno definirati `<style>` oznaku, te u tom slučaju se izgled nasljeđuje od nadređenog elementa. Dozvoljeno je unutar glavne `<template>` oznake definirati dodatne `<template>` oznake koje se mogu koristiti za repetitivno prikazivanje više elemenata. Primjer ovakvog korištenja će biti prikazan kasnije u tekstu. Definiranjem `<script>` oznaka se postavljaju funkcionalnosti sučelja i elemenata koje je korisnik kreirao u svome elementu. Korištenje korisnički definiranog elementa je prikazano u prilogu 3.2.

```
<hello-element name="World"></hello-element>
```

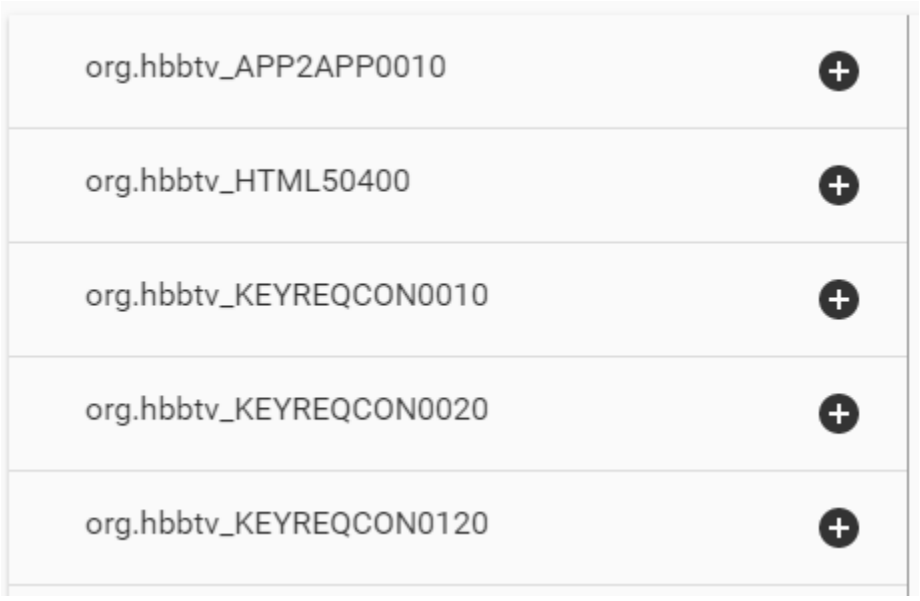
Prilog 3.2. Primjer korištenja korisnički definiranog elementa

Svaki element koji se koristi u Polymer-u je dostupan da se koristi unutar drugih elemenata, te je ponovna iskoristivost koda velika što povećava produktivnost i smanjuje vrijeme koje bi se u protivnom izgubilo na pisanje istih elemenata nekoliko puta. Prikazivanje elemenata koji su korišteni u izradi aplikacije bi bilo nepregledno te će primjeri koda biti dani u prilozima, također je bitno naglasiti da zbog prirode aplikacije sav kod neće biti dostupan u prilozima. Jedna bitna značajka Polymer biblioteke kod izrade korisničkog sučelja su smjernice grafičkog dizajna koje je Google razvio i implementirao u biblioteku. Zbog ovoga je grafičko sučelje puno ugodnije za korištenje i uklapa se u osam smjernica za razvoj kvalitetnog grafičkog sučelja koje omogućuje vrlo ugodno, jednostavno i intuitivno korištenje. U prilogu 3.3 je prikazan dio korištenja korisnički kreiranog elementa korištenjem Polymer biblioteke. Detaljnije objašnjenje će biti dano u nastavku teksta.

```
<div style="width: 35%">
  <iron-list id="testList" items="[[tests]]" selected-items="{{selectedItems}}"
selection-enabled multi-selection>
    <template>
      <div>
        <div tabindex$="[[tabIndex]]" class$="[[_computedClass(selected)]]">
          <div class="pad">
            <div class="primary">[[item.id]]</div>
          </div>
          <iron-icon icon$="[[iconForItem(selected)]]" class="check"></iron-icon>
        </div>
        <div class="border"></div>
      </div>
    </template>
  </iron-list>
</div>
```

Prilog 3.3 Primjer kreiranja liste elemenata

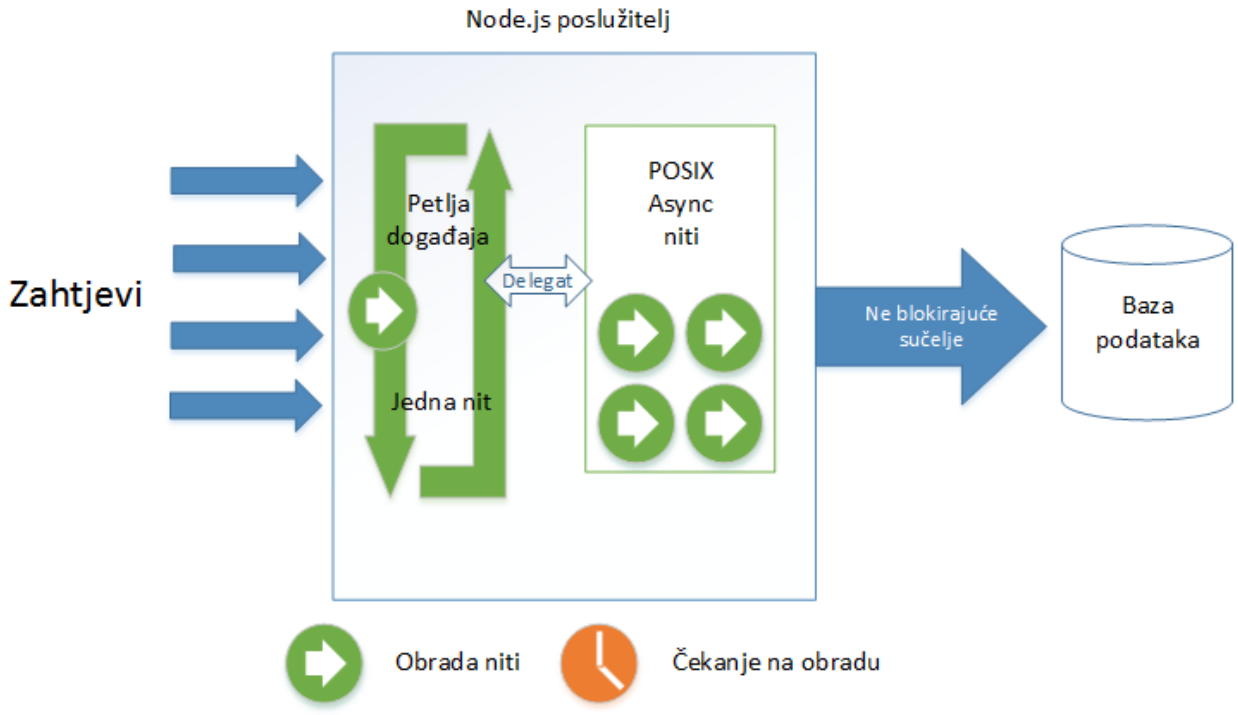
Na slici je primjer kako se kreira lista korištenjem ugrađenih elemenata iz biblioteke. Na ovaj način je moguće puno jednostavnije kreirati elemente i prikazati ih korisniku na način koji je u skladu s smjernicama za kreiranje kvalitetnog korisničkog sučelja. Na slici 3.4 je prikazan izgled sučelja koje je ostvareno kodom u prilogu 3.3. Vidljivo je da je uz relativno malu količinu pisanja koda moguće ostvariti vrlo dobre rezultate u korisničkom sučelju.



Slika 3.4. Prikaz rezultata korisničkog sučelja

3.3 Node.js server i SQLite baza podataka

Pri razvoju aplikacije je korišteno Node.js razvojno okruženje koje je namijenjeno razvoju aplikacija i programa koji se oslanjaju na korisnik-poslužitelj model funkcioniranja. Korištenje ovog razvojnog okruženja omogućuje programeru da na jednom računalu razvija i testira serverske komponente aplikacije bez potrebe za dodatnim uređajem. Node.js je kao i ostale biblioteke korištene za aplikacije bez potrebe za dodatnim uređajem. Poslužitelj kreiran korištenjem Node.js platforme se temelji na reakcijama na događaje i omogućuje asinkrone odgovore na te događaje. Zbog takve arhitekture nema blokiranja poslužitelja, te svaki zahtjev prema poslužitelju se sprema u petlju događaja i odgovor se šalje kada je zahtjev obrađen. U slučaju drugog zahtjeva prema poslužitelju, taj zahtjev ne čeka završetak obrade prijašnjeg zahtjeva nego također ulazi u petlju događaja i prima odgovor kada je taj zahtjev obrađen. To znači da server nije nedostupan za vrijeme obrade događaja. Prikaz rada ovakvog poslužitelja je prikazan na slici 3.5, te je vidljivo da nema bespotrebnog čekanja na obradu i gubitka vremena. Asinkronom obradom zahtjeva se smanjuje vrijeme potrebno za pristup bazi podataka.



Slika 3.5. Vizualni prikaz rada Node.js poslužitelja

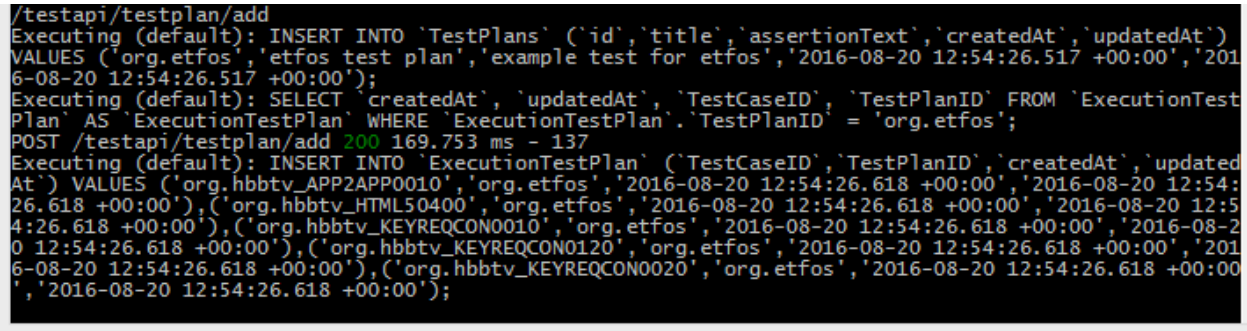
Svim funkcijama koje su na poslužitelju se pristupa korištenjem AJAX-a, što je skraćenica za Asinkroni JavaScript i XML. Na ovaj način se ostvaruje asinkrona priroda aplikacije i poslužitelja, te se resursi na poslužitelju ne troše bespotrebno. Zahtjevi prema poslužitelju se šalju samo kada je to potrebno i korisničko sučelje ne opterećuje poslužitelj bespotrebno. Neki zahtjevi su podešeni automatski što znači da se zahtjev generira sam u trenutku učitavanja stranice. Ovim putem se potrebni podatci prikazuju korisniku po učitavanju stranice te korisnik ne mora eksplicitno zatražiti navedene podatke. Svi zahtjevi su ostvareni korištenjem ugrađene funkcionalnosti u Polymer biblioteci koja pojednostavljuje generiranje i obradu ovih zahtjeva.

```
<iron-ajax
  content-type="application/json"
  id="write2DB"
  method="POST"
  url="/testapi/testplan/add"
  handle-as="json"
  on-response="handleResponse"
  debounce-duration="300">
</iron-ajax>
```

Prilog 3.6. Primjer AJAX zahtjeva korištenjem Polymer biblioteke

Da bi se generirao zahtjev potrebno je definirati kakav sadržaj će se slati i primati između klijenta i poslužitelja. Iz priloga 3.6 je vidljivo da je sadržaj koji će se slati (*content-type*) i sadržaj koji će se primati (*handle-as*) definiran kao JSON. JSON je standard koji koristi notaciju koja je čitljiva ljudima za slanje JavaScript objekata. AJAX koristi standardne HTTP metode za komunikaciju između poslužitelja i korisnika (*GET, POST, PUT, DELETE*) te se u ovom slučaju koristi *POST* metoda za slanje podataka prema poslužitelju. Navedena metoda se koristi zbog sigurnosti komunikacije te mogućnosti slanja gotovo neograničene količine podataka prema poslužitelju što su oboje zahtjevi za pravilan rad aplikacije. Unutar AJAX zahtjeva je naveden URL na koji poslužitelj šalje generirani zahtjev i čeka odgovor sa istog URL-a. Svaka putanja prema poslužitelju je povezana sa funkcijom koja obrađuje primljeni zahtjev te rezultat obrade šalje nazad klijentu. Na slici 3.7 je prikazano što se događa kada poslužitelj primi zahtjev i što se šalje kao rezultat obrade. U trenutku primanja zahtjeva se ispisuje putanja na koju je poslan

zahtjev i vrijeme koje je bilo potrebno da se zahtjev obradi, te se prikazuje što se događa, u ovom slučaju naredbe koje upisuju poslane podatke u bazu podataka.



Slika 3.7. Ispis u konzoli Node.js-a pri slanju AJAX zahtjeva

Primjer funkcije na poslužitelju koja dohvaća podatke iz baze podataka i šalje ih klijentu je dana u prilogu 3.8. Za rad s bazom podataka je korištena Sequelize biblioteka koja je napravljena za jednostavniji rad s objektno-relacijskim modelom baze podataka. Ovaj model olakšava rad s većim tablicama u bazi podataka zbog svojstva pretvaranja tablice u svojstva objekta sa pripadajućim metodama za rad s tim svojstvima. Korištenjem tog modela se tablice baze podataka mogu tretirati kao objekti u JavaScript-u što smanjuje količinu koda potrebnog za upisivanje i ispisivanje zapisa iz baze podataka.

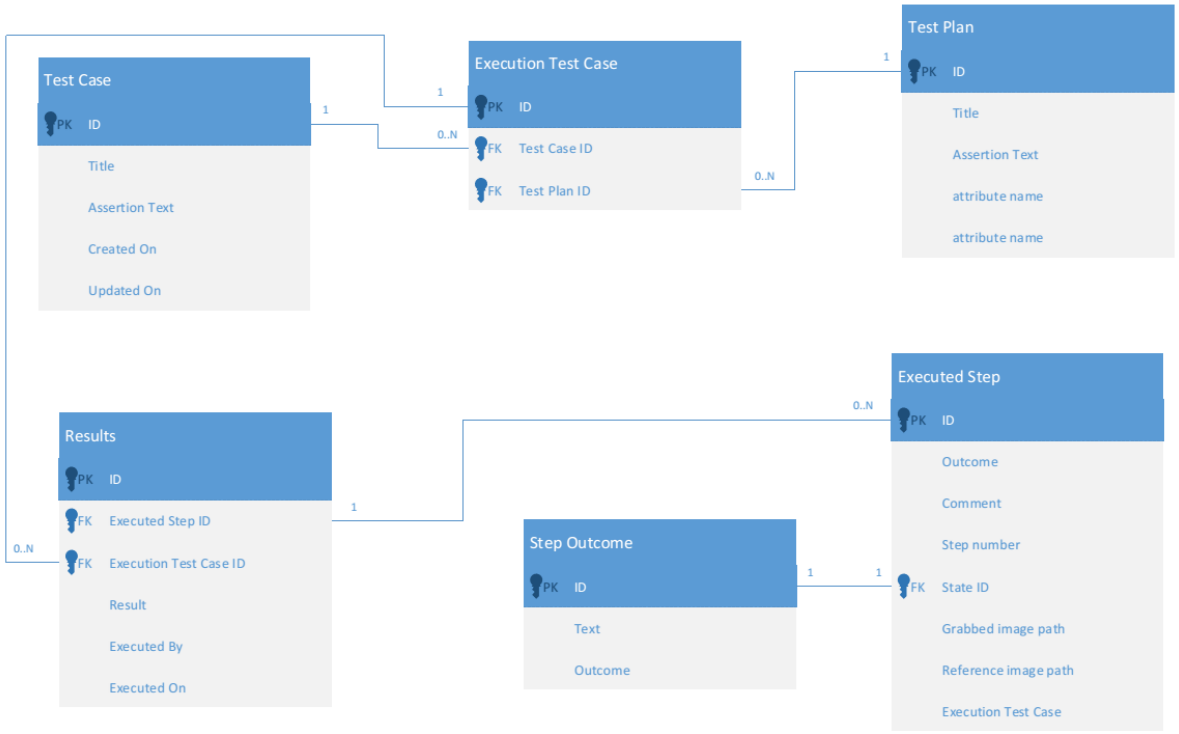
```
exports.getTests = function (req, res) {

    console.log(req.query.search);

    var search = req.query.search;
    models.Example.findAll(
        {
            order: [["id", "ASC"]],
            where: {
                id: {
                    $like: "%" + search + "%"
                }
            }
        }).then(function (example) {
        res.send(JSON.stringify(example));
    });
}
```

Prilog 3.8. Funkcija koja prima zahtjev od klijenta i vraća podatke u obliku JSON-a

Baza podataka odabrana za korištenje u aplikaciji je SQLite zbog svojstva da se ne oslanja na klijent poslužitelj model nego je ugrađena u krajnji proizvod. To znači da nije potreban poslužitelj koji u sebi čuva bazu podataka nego se ona nalazi lokalno kod korisnika. Na taj se način smanjuje opterećenje na poslužitelj, te je u krajnjem proizvodu imati poslužitelj standardnih performansi. Na slici 3.9 je prikazan model baze podataka koja je korištena u aplikaciji.



Slika 3.9. Model baze podataka korištene u aplikaciji

Baza podataka koja je korištena sadrži 6 tablica koje su sve međusobno povezane. Relacije između tablica u bazi su sljedeće:

- Svaki testni plan sadrži više testnih slučajeva
- Svaki testni slučaj pripada većem broju testnih planova
- Svaki testni slučaj ima više rezultata
- Svaki rezultat ima više izvršenih koraka
- Svaki izvršeni korak ima samo jedan rezultat

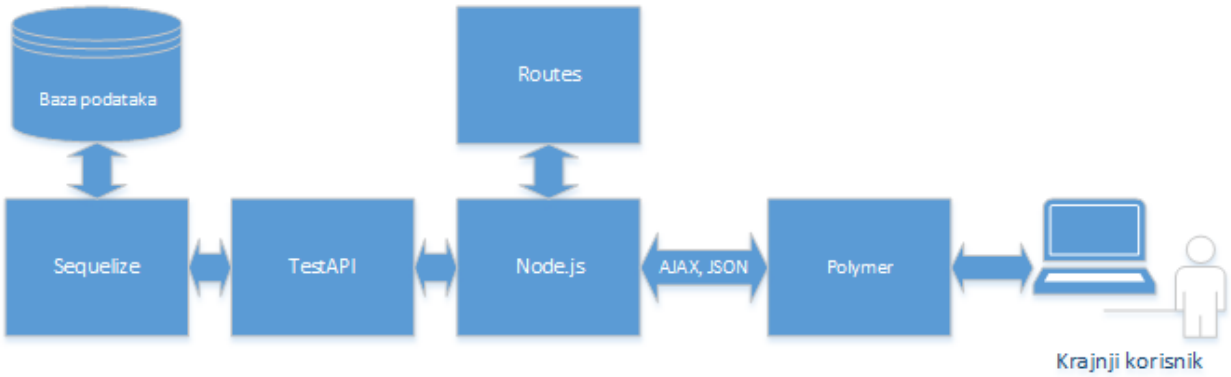
Zbog relacije između testnih planova i testnih slučajeva potrebna je spojna tablica koja omogućava stvaranje mnogo prema mnogo relacija. U suprotnom je nemoguće pravilno ostvariti takvu vrstu relacija. U spojnoj tablici se nalaze samo identifikatori testnih planova i testnih slučajeva koji pripadaju jedni drugima. Takva vrsta relacija se može promatrati kao primjer knjiga i pisaca. Svaki autor može napisati više knjiga i svaka knjiga može biti napisana od više pisaca. Ostale relacije koje se nalaze u bazi podataka su jednostavnije i mogu se ostvariti bez korištenja dodatnih tablica za povezivanje. Relacije gdje jedan testni slučaj ima više rezultata se ostvaruje upisivanjem identifikatora testnog slučaja u tablicu s rezultatima i svaki puta kada se upiše identifikator, upisuje se samo njemu svojstven rezultat. Primjer upisivanja testnog plana je dan na slici 3.7, te rezultat tog upisa u bazu podataka je dan na slici 3.10.

org.hbbtv_APP2APP0010	org.etfos
org.hbbtv_HTML50400	org.etfos
org.hbbtv_KEYREQCON0010	org.etfos
org.hbbtv_KEYREQCON0120	org.etfos
org.hbbtv_KEYREQCON0020	org.etfos

Slika 3.10. Primjer iz spojne tablice

Na slici je vidljivo kako jedan test plan imena „org.etfos“ je povezan s više testnih slučajeva koji se nalaze u posebnoj tablici, ali u tablici koja sadržava samo testne planove je pohranjen samo identifikator i opis tog testnog plana. Povezivanjem testnih planova i testnih slučajeva na ovaj način se ne troše bespotrebni resursi baze podataka, jer su spremljeni samo identifikatori koji jednoznačno predstavljaju svaki testni slučaj, plan i rezultat pojedinog testa.

4. Koncept rješenja



Slika 4.1 Blok dijagram elemenata korištenih u aplikaciji

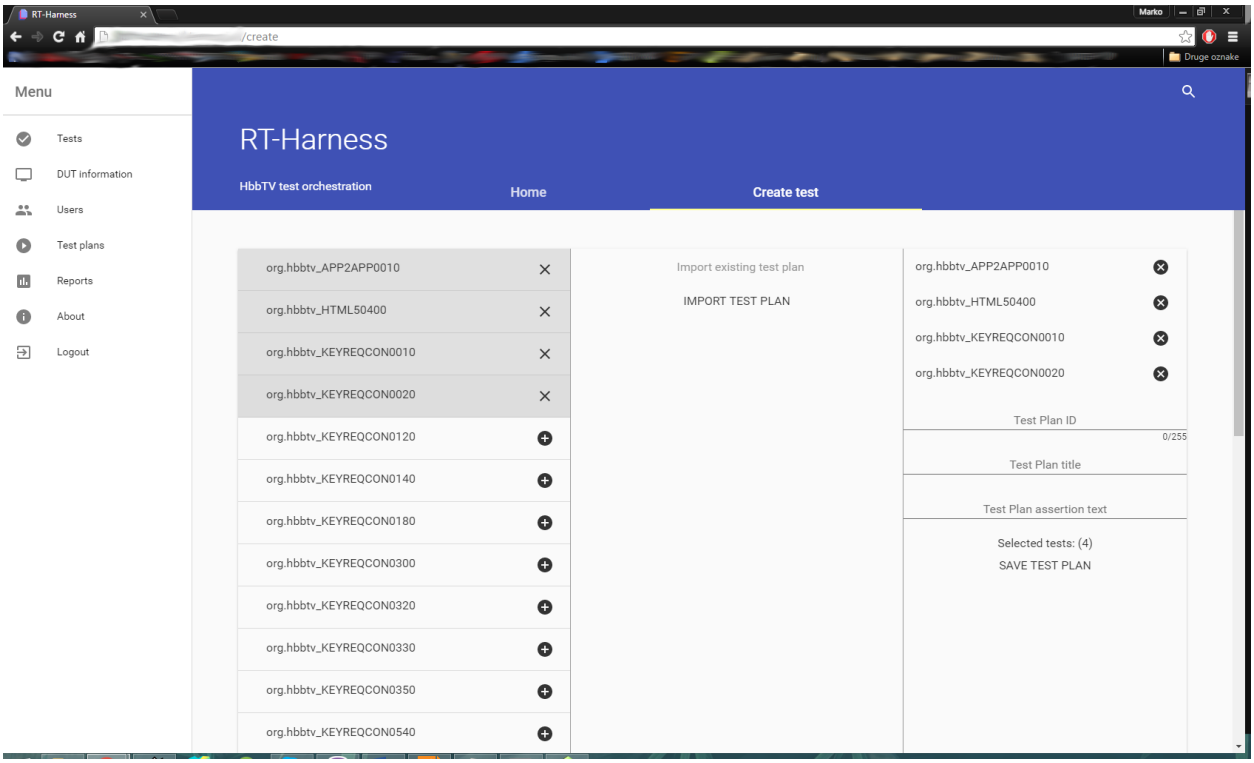
Na slici 4.1 su prikazani moduli koji su korišteni u aplikaciji, ali zbog samog opsega aplikacije i količine programskih datoteka na koje su moduli podijeljeni nije moguće prikazati kompletnu strukturu aplikacije. Broj datoteka koje se koriste u aplikaciji prelazi 60 000 zbog svih biblioteka koje služe kao potpora većim bibliotekama u programu, te je u dijagramu prikazana aproksimacija okruženja. Za ime aplikacije je odabran prefiks RT i definicija testnog okruženja dana od HbbTV grupacije, što čini RT-Harness

4.1 Opis modula aplikacije

4.1.1 Polymer

Polymer je korišten za stvaranje cjelokupnog korisničkog sučelja, te služi kao posrednik između korisnika i poslužitelja. Korisnik unosi naredbe u Polymer koji ih prosljeđuje poslužitelju, te poslužitelj ovisno o naredbama koje dobije od korisnika obrađuje informacije i prosljeđuje rezultat tih operacija korisničkom sučelju kako bih korisnik mogao reagirati ovisno o rezultatu koji je primio. Primjer kreiranja testnog plana je dan na slici 4.2 te je vidljivo kako su u jednu listu učitani svi dostupni testni slučajevi iz baze podataka. Testni slučajevi se ručno dodaju u direktoriji koji sadrži sve planove te u tom slučaju je potrebno osvježiti listu testnih slučajeva kako bi oni postali dostupni korisniku. Kreiranje novog testnog plana je ostvareno dinamičkim dodavanjem i uklanjanjem slučajeva u listu kako bi korisnik imao uvid koje testne slučajeve

odabire. Testni slučajevi koji su učitani u listu za spremanje u testni plan se mogu uklanjati iz liste koja sadrži planove koje je korisnik učitao i iz liste svih planova. Ova mogućnost je odabrana za kreiranje u svrhu smanjenja vremena potrebnog za dodavanje i uklanjanje testnih slučajeva.

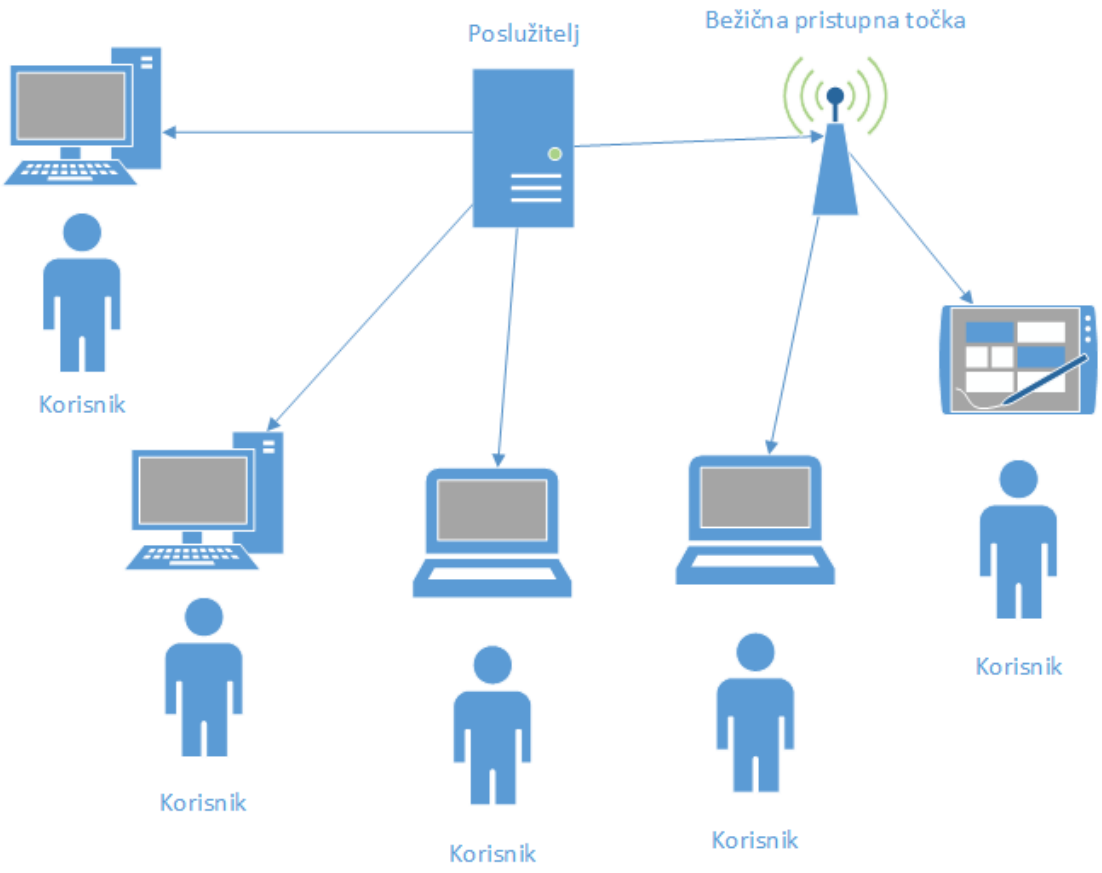


Slika 4.2. Izgled sučelja za kreiranje novih testnih planova

Polymer biblioteka je spremljena lokalno na razvojnom sustavu trenutno, ali u slučaju produkcijske verzije aplikacije moguće je zamijeniti lokalno spremanje biblioteke sa mrežnim servisima putem kojih je biblioteka dostupna. Ovo smanjuje količinu prostora koja aplikacija zauzima i samim time zahtjeve za pokretanje na računalu koje će biti korišteno za testiranje uređaja. Unutar sučelja za kreiranje novih testnih planova je moguće uvesti postojeći testni plan iz baze podataka, te se uvezeni plan može proširivati s dodatnim testnim slučajevima ili se slučajevi iz tog testnog plana mogu ukloniti i kreirati novi testni plan. Takvim pristupom se smanjuje opterećenje na korisnika ukoliko je potrebno provoditi isti testni plan na velikom broju uređaja. Sve moguće akcije korisnika su pokrivene u dijagramima korištenja i sučelje je pravljeno kako bi onemogućilo ljudsku pogrešku pri korištenju.

4.1.2 Node.js poslužitelj

Poslužitelj služi kako bi se sve komponente aplikacije mogle razviti na jednom računalu, te za prikazivanje ponašanja poslužitelja i aplikacije u specifičnim slučajevima. U produkcijskoj verziji aplikacije moguća je centralizirana arhitektura s jednim poslužiteljem na koji se povezuje veći broj korisnika. Na taj način bi svi testni slučajevi bili sinkronizirani na jednom mjestu, zajedno sa svim rezultatima i testnim planovima. Centraliziranjem se rješava problem praćenja verzija testnih planova i pregledavanje rezultata testiranja. Svojstvo centralnog upravljanja dobiva sve veći značaj sa povećanjem broja korisnika. Niti jedan korisnik nema direktan pristup bazi podataka, nego korištenjem sučelja između Polymer biblioteke i Node.js poslužitelja, te su sve funkcije na poslužitelju zaštićene od krivih unosa ili mogućih nemoralnih namjera poput mijenjanja rezultata testova u svrhu dobivanja prolazne ocjene, pristup testnim slučajevima i sličnim akcijama.



Slika 4.3. Prikaz centralizirane strukture

Na slici 4.3 je prikaz centralizirane strukture aplikacije koja omogućuje istovremeno korištenje od strane nekoliko korisnika bez ometanja u radu ostalih korisnika, što znači da je ovakva struktura vrlo pogodna za veliki broj korisnika istovremeno. Poslužitelj na raspolaganju ima velik broj datoteka gdje su zapisane putanje koje su dozvoljene korisniku da posjećuje i pripadajuće funkcije koje će obaviti unaprijed definiranu zadaću. Datoteka koja sadrži sve putanje i pripadajuće funkcije također definira i metodu komunikacije između poslužitelja i klijenta. Razlikovanje korisnika u slučaju centralizirane mreže i višestrukog pristupa se ostvaruje korištenjem jedinstvenog identifikatora za svakog korisnika. Na taj način se ostvaruje stvarna nezavisnost višestrukih instanci aplikacije i rad na više terminala je moguć bez ikakvih poteškoća.

4.1.3 TestAPI datoteka

Datoteka u kojoj su zapisane sve funkcije poslužitelja koje odgovaraju na korisničke zahtjeve se u ovom slučaju naziva testAPI datoteka. Datoteka sadržava velik broj funkcija koje pokrivaju sve slučajeve koji su korisniku dopušteni. Bilo kakav pokušaj ručnog upisivanja nepostojeće adrese prema poslužitelju će biti ignoriran od strane servera, dok će aplikacija prikazati prazno sučelje. Zbog pretpostavke da je nepostojeća adresa greškom upisana, poslužitelj neće poduzeti nikakvu protumjeru prema korisniku. U prilogu 4.4 je dan primjer putanje i pripadajuće funkcije na koju se šalje AJAX zahtjev za kreiranje novog testnog plana. Dolaskom zahtjeva na tu putanju primjenom primjerene metode, zahtjev se prosljeđuje funkciji koja prima tijelo zahtjeva i obrađuje ga.

```
router.post('/testplan/add', controllers.addTestPlan);
```

Prilog 4.4. Putanja sa pripadajućom funkcijom i definiranom POST metodom

Funkcija *addTestPlan* izvršava kreiranje testnog plana. U tijelu zahtjeva se nalaze identifikatori testnih slučajeva koje je korisnik odabrao, te identifikator i podatci o testnom planu koje je korisnik definirao u obliku JSON objekta. Primljeni objekt se parsira u oblik prigodan za rad sa Sequelize bibliotekom koja obavlja upisivanje testnog plana sa pripadajućim testnim slučajevima. Nakon obrade zahtjeva rezultat obrade se šalje u korisničko sučelje, te logika

sučelja odlučuje o sljedećoj akciji s primljenim rezultatom. Prilog 4.5 prikazuje opisanu funkciju i upis u bazu podataka.

```
exports.addTestPlan = function (req, res) {

    var tp = {validTC: []};
    var success = [];

    for (var i = 0; i < req.body.plans.length; i++) {
        success.push(req.body.plans[i].id);
    }
    tp.validTC.push(success);

    models.TestPlan.create({
        id: req.body.id,
        title: req.body.title,
        assertionText: req.body.text
    }).then(
        function (tp1) {
            tp1.setTestCase(success);
        }).then(function () {
        res.setHeader('Content-Type', 'application/json');
        res.send(JSON.stringify(tp));
    });
}
```

Prilog 4.5. Kreiranje novog testnog plana i upisivanje u bazu podataka

4.2 Testiranje aplikacije

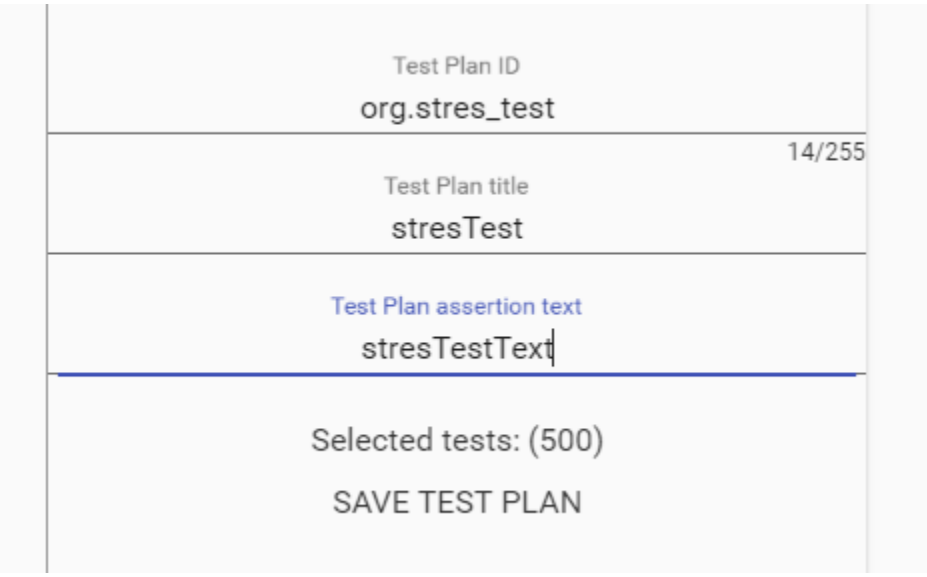
Testiranje aplikacije se obavlja kako bi se utvrdila ispravnost aplikacije i rad svih modula koji su navedeni i napisani. Testiranje treba biti što opsežnije kako se ne bi pojavili problemi kada aplikacija bude isporučena korisniku. Postoji standardna praksa koja se koristi pri testiranju web aplikacija i uključuje sljedeće komponente:

- Testiranje funkcionalnosti aplikacije
- Testiranje upotrebljivosti
- Testiranje sučelja
- Testiranje kompatibilnosti
- Testiranje performansi
- Testiranje sigurnosti

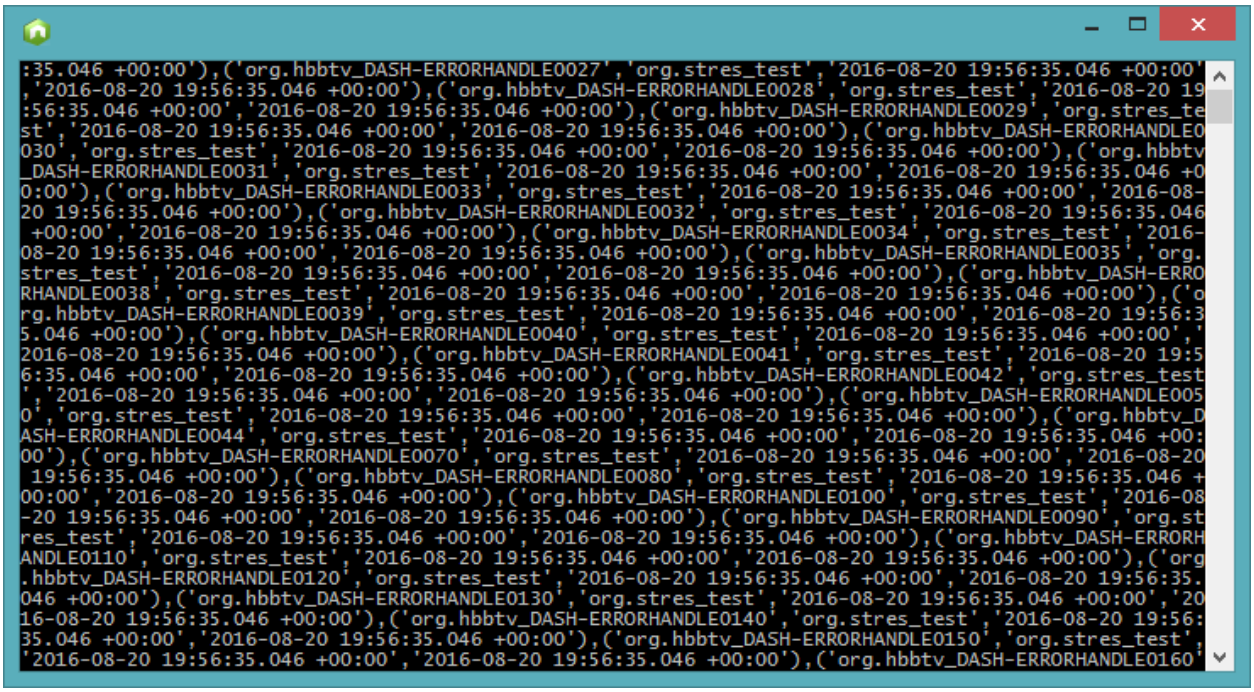
Testiranje funkcionalnosti se obavlja ispitivanjem ispravnosti svih putanja koje su definirane, povezanost s bazom podataka, te ispitivanje formi za popunjavanje obrazaca, u ovom slučaju kreiranje testnih planova. Ispituju se odlazne i dolazne veze unutar korisničkog sučelja. Upotrebljivost aplikacije se ocjenjuje jednostavnošću korištenja aplikacije, kretanjem kroz izbornike, jasnoćom danih uputa za korištenje i konzistentnošću kroz cijelu aplikaciju. Bitno je da glavno upravljačko sučelje bude uvijek dostupno bez obzira na lokaciju unutar aplikacije. Testiranje sučelja se u ovom slučaju odnosi na testiranje sučelja između poslužitelja i baze podataka. Bitno je provjeriti da nema nikakvih pogrešaka u komunikaciji između te dvije komponente aplikacije, zbog velike važnosti baze podataka. Testiranje kompatibilnosti je vrlo važno zbog mogućnosti da korisnik aplikaciju koristi na velikom broju različitih web preglednika, operativnih sustava i mobilnih uređaja. Velika važnost se posvećuje kompatibilnosti s različitim web preglednicima zbog različite implementacije određenih standarda. Performanse aplikacije se trebaju testirati na način da velik broj korisnika istovremeno pristupa aplikaciji. Još jedan način testiranja performansi je stres testiranje gdje se sustav i aplikacija pokušavaju upotrijebiti van specifikacije. Na taj način se promatra reakcija sustava na takvo opterećenje i u slučaju pada (eng. *crahs*) se promatra kako se sustav oporavlja od takvog stanja. Sigurno testiranje se vrši pokušajima unošenja direktnih adresa u adresnu traku preglednika bez prethodne prijave, u kojem slučaju se stranica ne bi trebala prikazati. Aplikacija treba odbiti bilo kakve netočne unose u polja za unos od strane korisnika. Bilo kakvi pokušaji probijanja sigurnosti sustava se trebaju zabilježiti interno na poslužitelju kako bi upravitelji sustava imali uvid u sigurnosne prijetnje.

Aplikacija je testirana po svim točkama i nije došlo do nikakvog očitog problema. U slučaju kada dva korisnika pokušaju istovremeno kreirati testni plan istog imena, poslužitelj će odbiti jednog korisnika, bez očitog statističkog uzorka. Aplikacija je testirana na raznim web preglednicima i nije došlo do nikakvih problema za vrijeme testiranja. Sva polja koja su dostupna korisniku na popunjavanje su zaštićena na način da se poslužitelju šalje samo tekst iz polja i nije moguće poslati nikakav dokument ili malicioznu skriptu. Za vrijeme testiranja performansi nije došlo do rušenja sustava, te zbog toga nije bilo moguće promatrati oporavak sustava od pada. Pretpostavka je da će sustav pokušati ponovno učitati potrebne dokumente nakon prestanka prevelikog opterećenja. Stabilnost baze podataka je testirana na način da su u jedan testni plan bili učitani svi dostupni testni slučajevi (500) i nakon toga poslani i upisani u

bazu. Za vrijeme testiranja nije se manifestiralo nikakvo neuobičajeno ponašanje i testni plan je uspješno kreiran. Pogled na upis svih 500 planova u bazu je dan na slici 4.6 i slici 4.7.



Slika 4.6. Primjer učitavanja svih dostupnih testnih slučajeva u jedan testni plan



Slika 4.7. Primjer ispisa poslužitelja za vrijeme stres testa

5. Zaključak

Aplikacija je rađena u svrhu povezivanja sustava za automatsko testiranje uređaja koji trebaju zadovoljiti HbbTV 2.0 specifikaciju. Zbog količine testova koje je potrebno izvršiti na uređaju pojavila se potreba za automatizacijom procesa testiranja. Aplikacija uspješno kreira testne planove proizvoljne veličine i kompleksnosti testova. Korisniku je trebalo omogućiti što lakše i jednostavnije korištenje uz privlačan izgled korisničkog sučelja. Bitno svojstvo je bila brzina rada aplikacije i upisivanja i dohvaćanja informacija iz baze podataka. Za vrijeme izrade aplikacije došlo je nekoliko promjena u zahtjevima, posebice u sučelju za automatsko testiranje. Aplikacija je ispunila sve zahtjeve, kreiranja i izvršavanje testnih planova. Zbog količine zadataka i problema koje je trebalo riješiti nedostaju određena svojstva aplikacije koja je moguće dodati bez previše istraživanja. Velik problem je predstavljalo pravilno ostvarivanje komunikacije između svih modula zbog količine biblioteka i datoteka koje su bile uključene u projekt. Baza podataka za kreiranje testnih planova i spremanje rezultata je bila nepostojeća i bilo ju je potrebno modelirati prema zahtjevima aplikacije imajući u vidu sve zahtjeve i količinu mogućih korisnika. Otegotna okolnost je bila Polymer biblioteka koja je još u fazi razvoja i mnoga ažuriranja su mijenjala neka od svojstava na koje se oslanjalo prije dolaska ažuriranja. Trenutno je aplikacija još u fazi razvoja, ali osnovna ideja i glavne funkcionalnosti su ostvarene.

Literatura

[1]. https://www.w3.org/2013/10/tv-workshop/papers/webtv4_submission_10.pdf [pristup ostvaren 15.6.2016.]

[2]. https://www.hbbtv.org/wp-content/uploads/2015/07/HbbTV_specification_2_0.pdf [pristup ostvaren 15.6.2016.]

[3]. http://www.softwaretestinghelp.com/web-application-testing/ [pristup ostvaren 15.8.2016.]

Sažetak

U diplomskom radu *Sprega HbbTV testnog sustava sa sustavom za automatsko testiranje* opisuje se izrada aplikacije za automatsko testiranje uređaja koji trebaju zadovoljiti HbbTV standard u svrhu plasiranja uređaja na tržište. Proizvođači žele ispuniti zahtjeve navedenog standarda kako bi bili kompetentni na tržištu zbog sve većeg prisustva pametnih (eng. *Smart*) televizijskih prijemnika.

HbbTV standard je još uvijek u fazi razvoja, ali su ga neke europske države već implementirale i to otvara mogućnosti proizvođačima da u trenutku šireg prihvaćanja standarda ponude svoj uređaj kao kompatibilan. Za izradu aplikacije su korišteni besplatni alati kao jedan od zahtjeva i dokaza da je moguće izraditi aplikaciju koja zadovoljava standarde svjetskih korporacija i standardizacijskih tijela. Neke od biblioteka i razvojnih okruženja korištenih za izradu ovog rada su Node.js, Sequelize, Polymer-project, SQLite te još mnogi drugi. Kao razvojno okruženje je korišten WebStorm, koji pojednostavljuje pisanje koda svojim funkcionalnostima. Aplikacija je uspješno kreirana, ostvarena je centralizirana arhitektura upravljanja sa distribuiranom mrežom korisnika. Na taj način jedan poslužitelj može imati velik broj korisnika, te je upravljanje testnim slučajevima koji dolaze od standardizacijskog tijela uvelike pojednostavljen.

Ključne riječi: HbbTV, Node.js, Polymer, RT-Harness, Sequelize, SQLite, database, testni plan, testni slučaj, razvoj aplikacije, pametni TV

Summary

The thesis *Coupling HbbTV test system with automatic testing subsystem* describes the development of application for automatic testing devices that should meet the HbbTV standard for the purpose of placing the device on the market. Manufacturers want to meet the requirements of the above standards in order to be competent in the market due to an increasing number of smart television receivers.

HbbTV standard is still under development, but some European countries have already implemented it and it offers possibilities for manufacturers to offer the device as compatible in moment of wider acceptance. To create this application free tools were used as one of the demands and proof that it is possible to develop an application that meets the standards of international corporations and standardization bodies. Some of the libraries and development environments used for the preparation of this work are Node.js, Sequelize, Polymer-project, SQLite and many others. As a development environment WebStorm is used, which simplifies writing code with its functionality. The application is successfully created, realized the centralized management architecture with distributed network of users. In this way, one server can have a large number of users and the management of test cases that come from standardization bodies is greatly simplified.

Keywords: HbbTV, Node.js, Polymer, RT-Harness, Sequelize, SQLite, database, test plan, test case, application development, smart TV

Životopis

Marko Plenković je rođen u Vranovcima 3.ožujka 1993. godine. Svih osam razreda osnovne škole završava odličnim uspjehom te je '07. sudjelovao na županijskom natjecanju iz fizike. Nakon završene osnovne škole, upisuje Tehničku školu u Slavonskom Brodu, smjer Tehničar za mehatroniku. Sve razrede prolazi s odličnim i tokom školske godine 2009./2010. sudjeluje na županijskom natjecanju iz engleskog jezika. Elektrotehnički fakultet upisuje 2011. godine te svaku godinu redovno završava. Godine 2014. završava preddiplomski studijski program Komunikacija i informatike na Elektrotehničkom fakultetu u Osijeku, te upisuje diplomski studijski program. Na petoj godini studija potpisuje ugovor o stipendiranju s tvrtkom RT-RK.