Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matjaž Jurečič

Sistem za dinamično upravljanje izgleda klienta IPTV

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Franc Solina

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Zasnujte in razvijte rešitev za ločeno upravljanje grafičnega vmesnika klienta za IP televizijo, kar naj bi omogočilo hitre spremembe izgleda uporabniškega vmesnika.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matjaž Jurečič, z vpisno številko **63070239**, sem avtor diplomskega dela z naslovom:

Sistem za dinamično upravljanje izgleda klienta IPTV

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Franca Soline,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 27. maj 2015 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
SVG	scalable vector graphics	skalabilna vektorska grafika
STB	set top box	TV komunikator
CSS	cascading style sheets	prekrivni slog
DBDD	develop, build, deploy, debug	razvij, zgradi, uvedi, razhrošči
JSON	JavaScript object notation	zapis JavaScript objekta
MVC	model view control	model pogled kontrolnik
WAR	web aplication archive	
CDN	content delivery network	omrežje za dostavljanje vsebin

Povzetek

V diplomski nalogi je predstavljena rešitev za ločen razvoj prezentacijskega dela aplikacije od logike. Cilj sistema je napraviti dinamičen sistem za upravljanje grafičnega vmesnika, ki bi olajšal prilagoditev klienta ter zmanjšal čas potreben za spremembo obstoječega ali razvoj novega izgleda. Predstavljeni so problemi, ki so bili povod za razvoj sistema, tehnologije, ki smo jih uporabili za razvoj, končna arhitekturna rešitev in razvita orodja.

Ključne besede: urejevalnik izgleda, urejevalnik predlog, node.js.

Abstract

A system for dynamic management of the presentation layer of an IPTV client

The work presents an application for development of the presentation layer of an IPTV system. The goal was to create a dynamic solution for editing the look and feel of a graphical user interface that would ease the customisation of the client and reduce the time necessary to change existing UI or create a new one. The document also presents the problems that led to the development of the system, technologies that were used, how the architecture was designed and which tools were created.

Keywords: theme editor, presentation layer parsing, node.js.

Poglavje 1

Uvod

Rešitve IPTV so veliki, vseskozi razvijajoči se sistemi, ki servirajo televizijski program mnogo odjemalcem. Gre za kompleksen sistem, sestavljen iz množice različnih strežnikov, podakovnih baz in predstavitvenega dela (klienta), kateri teče na TV komunikatorjih (STB), mobilnih napravah, spletnih brskalnikih in tudi nativno na nekaterih pametnih televizorjih. V diplomskem delu se bomo osredotočili na klienta.

Izgled aplikacije je definiran s SVG in CSS in je v celoti odvisen od javascript middlewarea ki teče na komunikatorju. To v praksi pomeni, da je za vsak videz narejena nova verzija teh CSS in SVG datotek (katere vsebujejo tudi precej javascript kode), z malenkostnimi spremembami, ki definirajo barve in postavitve elementov. To privede do velikega podvajanja kode, katera mora biti, zaradi tesne povezanosti z domensko logiko, ob vsaki spremembi na strežniku tudi ustrezno posodobljena.

Dodatno smo želeli uvesti možnost, da lahko neodvisno spreminjamo tudi samo delovanje grafičnega vmesnika. Do sedaj smo lahko spreminjali le izgled, funkcionalnosti in delovanje pa je diktiral middleware in je bilo za vse naročnike enako. Cilj je bil, da naredimo bolj dinamičen in prilagodljiv način, ki bi omogočal popolno prilagoditev grafičnega vmesnika in v zameno za sicer nekoliko daljši razvojni cikel novega videza (če se kupec odloči, da želi tudi poosebljeno funkcionalnost), korenito zmanjša čas potreben za vzdrževanje

in testiranje ob izdaji novih verzij middlewarea.

To smo dosegli tako, da smo popolnoma ločili logiko od predstavitve, ter dodali nov sloj v naš sistem. Grafični vmesnik sedaj teče na Node.js strežniku, ki sedaj služi kot vmesnik med strežnikom in klientom. Razvili smo tudi orodja za prilagajanje izgleda, ki tečejo v spletnem brskalniku in omogočajo enostavno spreminjanje, ki ga lahko izvajajo kar naročniki sami. Delovanje grafičnega vmesnika pa je v določeni strukturi definirano v json datotekah.

V drugem poglavju so na kratko predstavljeni problemi, ki smo jih želeli rešiti. V tretjem poglavju so na kratko predstavljene tehnologije, ki smo jih uporabili pri svoji rešitvi. V četrtem poglavju predstavim urejevalnik izgleda, na koncu pa še opišem dosedanji razvoj in trenutno stanje rešitve.

Poglavje 2

Problem

Klient je edini del celotnega sistema, ki se prilagaja kupcu. Vsak kupec ima možnost, da dobi poosebljen izgled vmesnika IP televizije in skupaj z rastjo uporabe našega sistema, smo začeli opažati, da gre za ta del veliko programerskega časa. Še iz časov, ko smo imeli zelo malo različnih izgledov vmesnika IPTV, je bila naša koda zelo tesno povezana z delovanjem samega middlewarea. Ob vsaki spremembi je bilo potrebno posodobiti vse poosebljene izglede, saj so posamezni odseki zaradi pretesne povezanosti prenehali delovati. V večini primerov je bilo potrebno popraviti iste odseke posameznih predlog, kar pomeni veliko nezahtevnega in monotonega dela ter slabo porabljen čas razvijalcev.

Tesna povezanost ločenih komponent je prav tako botrovala mnogo regresijam, ki jih je bilo zaradi narave sistema zelo težko odkriti. Pred objavo nove verzije je bilo potrebno pregledati vsakega od izgledov, saj so se med seboj (večinoma) razlikovali dovolj, da enostavno kopiranje popravkov ni zagotavljalo pravega delovanja. Prav tako je bilo potrebno popravljati izglede, ki so delovali enako, a so imeli drugačne ikonice, barve in slike. Dobro ločena logika od predstavitve bi tukaj prihranila veliko časa, še posebej, ko v enačbo vključimo različne komunikatorje in razlike v delovanju med njimi.

Za odpravo teh pomanjkljivosti smo se odločili za popolno prenovitev predstavitvenega dela. Poleg reševanja zgoraj navedenih problemov smo

prav tako želeli omogočiti večjo fleksibilnost predstavitvenega dela, ki bi omogočala popolnoma različne izkušnje pri uporabi našega sistema. Nove poosebljene rešitve ne bi več zajemale zgolj površinskega izgleda, temveč bi lahko posamezni kupci sistema popolnoma predrugačili delovanje vmesnika. Velik del poosebljanja bi se preko razvitih orodij prestavil na same operaterje IPTV, kar bi dodatno sprostilo čas našim programerjem in hkrati dalo proste roke ponudnikom storitev IPTV glede izgleda in delovanja njihove storitve.

Poglavje 3

Raziskovanje in načrtovanje

Problem je zelo kompleksen, ker mora biti novi sistem združljiv z veliko že obstoječe kode, zato je bilo potrebno vsako stvar temeljito raziskati in razčleniti. Prav tako smo že kmalu ugotovili, da smo zagrizli v veliko večji problem, kot smo na začetku predvidevali, hkrati pa smo tudi dodajali nove storitve k osnovni ideji, zato smo se odločili, da bomo problem razstavili ter se ga lotili po korakih.

V prvem delu smo se odločili napraviti urejevalnik predlog na obstoječem klientu. Tu notri so všteti:

- specifikacija za GUI,
- spletni urejevalnik,
- sistem za dinamično serviranje predlog SVG,
- OpenAPI metode za deljenje konfiguracij predlog SVG,
- prototip predloge na novem sistemu.

Že za ta del je bilo potrebno spremeniti precejšen del kode klienta, toda šele v drugem delu bi se naredial popolna preureditev obstoječe kode, podprl bi spreminjanje delovanja grafičnega vmesnika s konfiguracijo itd.

Nato je bilo potrebno določiti delovanje samega sistema. V prvi fazi smo raziskovali tehnologije, ki so nam bile na voljo skupaj z različnimi koncepti

definiranja uporabniških vmesnikov. Nato smo za vsako idejo izvedli preizkus. Če je bil preizkus neuspešen, smo idejo takoj zavrgli, izmed uspešnih pa smo določili smer razvoja.

Na koncu smo se odločili, da bo za GUI skrbel node.js strežnik, za izgled se bo uporabljala lastna notacija v formatu JSON, katerega bo naš spletni urejevalnik razčlenjeval in na podlagi njega omogočal enostavne spremembe, ki se bodo odražale neposredno v svg datotekah. Potem smo definirali še način dela, saj je le to moralo zaradi omejenega števila razvijalcev potekati karseda hitro. Z avtomatskimi Maven/Grunt skriptami smo vzpostavili DBDD način razvijanja.

3.1 Node.js

Za prezentacijske datoteke smo uporabili ločen Node.js strežnik, ki teče na lokalnem omrežju. Razvili smo program, ki teče na njem in služi kot nekakšen vmesnik med domensko logiko in prezentacijo na klientu. Ta strežnik servira in deli prave datoteke GUI, ter jih hkrati tudi razčlenjuje in pripravlja. Na njem teče tudi spletni urednik.

Node.js je programska platforma za razširljive spletne aplikacije ter nasploh aplikacije, ki tečejo na strežniku. Aplikacije za Node.js so napisane v jeziku JavaScript in lahko tečejo znotraj Node.js sistema na Windows, Max OS X in Linux platformah v enaki obliki [2].

Glede na to, da celoten klient sloni na JavaScriptu, ki se potem izvaja na posebnem brskalniku znotraj TV komunikatorja in neobčutljivosti na platformo, je bil Node.js naravna izbira. Poleg tega nam omogoča izvajanje testiranja kode klienta, saj tega na našem brskalniku ne omogoča nobeno od ogrodij za testiranje. Zaradi izkušenj in podrobnega poznavanja JavaScript-a ter ogrodij, kot je naprimer jQuery, je bil prehod dokaj naraven.

3.2 Regularni izrazi

Za razčlenjevanje opisa grafičnega vmesnika smo uporabili izraze regex. Regex je zaporedje znakov, ki tvorijo vzorec, s katerim preverjamo ujemanje nizov, iščemo po nizih, itd. [3]. Glede na to, da smo vpeljali lasten zapis podatkov, ki definirajo izgled in delovanje grafičnega vmesnika, smo morali te podatke nekako razčleniti ter določiti njihovo pravilnost. Za to smo uporablili izraze regex, saj z dobro napisanimi izrazi lahko zagotovimo robustnost samega interpreterja in ga ne zmoti več vsak podvojen presledek in druge stvari, ki jih ljudje radi vnašamo v svojo kodo.

Ogromno jezikov podpira izraze regex, ki pa se med seboj nekoliko razlikujejo [1]. V jedru so si sicer vse implementacije zelo podobne, imajo pa svoje značilnosti ter različne funkcije, ki omogočajo bolj kompleksne primerjave in iskanje oziroma veliko enostavnejšo sintakso za zahtevnejše probleme. Mi smo uporabljali JavaScript implementacijo regexa.

Primer 3.1 *Enostaven izraz regex*

```
var regex = /\w*\d+/
```

Enostavni regex iz primera (3.1) uporabimo za testiranje tekstovnih nizov. `\w` se ujema z vsako besedo, `\d` pa z enim numeričnim znakom. Besedo v regexu sestavlja niz alfanumeričnih ali numeričnih znakov, ter podčrtaj [4]. Če nato uporabimo `našeBesedilo.match(patt)` dobimo naslednje rezultate:

- "nekaj besed" ->se ne ujema z vzorcem
- "nekaj besed in potem 44" ->se ujema, prva vrednost v tabeli zadetkov je število 44
- "44" ->se ujema, prva vrednost v tabeli zadetkov je število 44

Izraz iz primera (3.1) torej vsebuje vzorec za niz sestavljen iz `0-n` besed katerim sledi število. Ukaz `match` potem preveri ali se naš niz ujema z vzorcem v izrazu regex in v primeru, da se ujema, število shrani na prvo mesto

tabele, ki jo vrne. Vzorec je popolnoma neodvisen od števila besed, ki se nahajajo pred številko. S kompleksnejšimi izrazi lahko naredimo zelo robusten validator in parser.

3.3 SVG

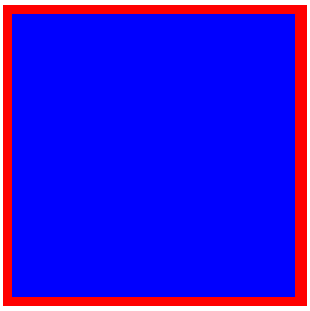
SVG je kratica za Scalable Vector Graphics, ki predstavlja format za vektorsko grafiko, ki temelji na XMLju in podpira dvodimenzionalno risanje s podporo za animacije in interakcije [5]. SVG je odprti standard, ki ga že od leta 1999 razvija W3C. Naš klient uporablja SVG namesto HTMLja, ker imajo komunikatorji boljšo podporo za to tehnologijo in SVG na njih deluje precej hitrejše. Stvari se sicer spreminjajo in v prihodnosti želimo podpreti tudi HTML5, toda za zdaj na nekaterih strojno zelo podhranjenih komunikatorjih to ni mogoče.

SVG pozna tudi poenostavljene profile, ki vsebujejo zmanjšan nabor funkcij SVG Basic in SVG Tiny. Naš brskalnik podpira SVG Tiny specifikacijo, ki pa jo razširja z nekaterimi svojimi rešitvami. Prav tako so nekatere vektorske operacije strojno podprte na samih komunikatorjih in so strojno pospešene.

Primer 3.2 *Enostaven SVG dokument*

```
<svg width="1280" height="720">
  <rect width="100" height="100" style="fill:rgb(0,0,255);
    stroke-width:3;stroke:rgb(255,0,0)">
</svg>
```

Primer (3.2) prikazuje enostaven SVG dokument, ki izriše pravokotnik velikosti 100 krat 100. Ta velikost je relativna in je odvisna od starša. V tem primeru, kjer nad starševskim elementom ni nobene transformacije, se bo izrisal kvadrat z dolžino stranice 100 pikslov. V primeru vidite tudi lastnost `style`, ki jo verjetno poznate iz spletnega oblikovanja, seveda lahko uporabimo tudi CSS.



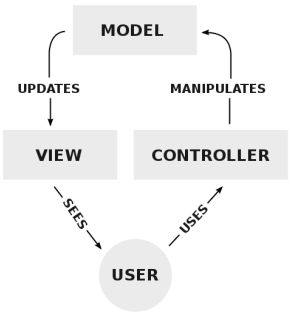
Slika 3.1: Izris SVG dokumenta iz primera (3.2)

Slika 3.2: Kompleksnejša vektorska risba

Primer (3.2) in slika 3.1 predstavljata zelo enostaven primer, ki ne pokaže zmogljivosti vektorske grafike in formata SVG. Na sliki 3.2 lahko vidite nekoliko bolj kompleksen primer, ki je dostopen tudi na internetu [6].

3.4 Grunt

Grunt je orodje ki omogoča avtomatizacijo procesov pri delu z JavaScriptom in CSSjem [7]. S pomočjo Grunta se lahko izognemo ponavljajočim procesom, kot so optimizacija slik, testiranje kode, minimizacija datotek CSS in podobno. Grunt teče na platformi Node.js. Z enostavno konfiguracijsko da-



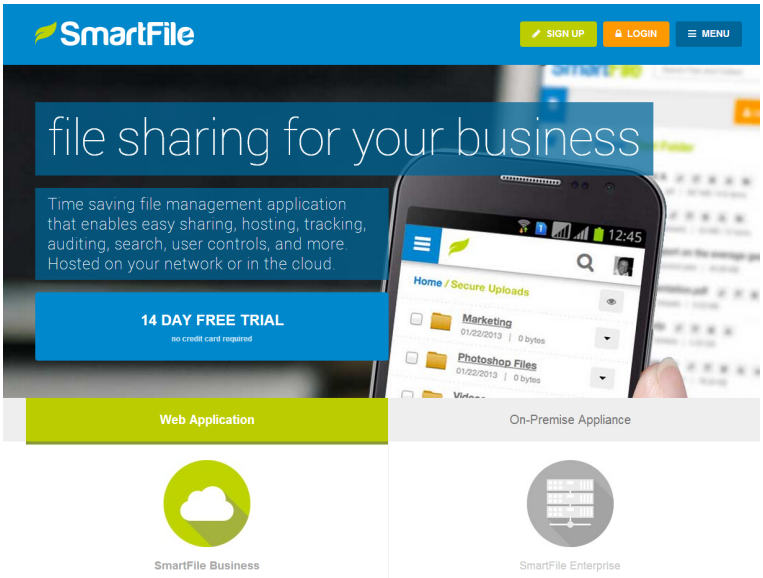
Slika 3.3: Prikaz arhitekturnega modela MVC

toteko potem nastavimo knjižnice, ki jih uporablja naš projekt, vrstni red nalaganja skript, testiranje kode itd. Z gruntom smo prihranili precej časa in se ognili težavam z usklajevanjem knjižnic med člani ekipe.

3.5 AngularJS

AngularJS je odprtokodno ogrodje za spletne aplikacije, ki nam olajša izdelavo single-page aplikacij [9]. To so spletne aplikacije, ki izgledajo ter se obnašajo enako kot namizne aplikacije. Vse poteka znotraj ene strani, menja pa se le vsebina preko asinhronih klicev.

AngularJS nam olajša tudi delo v arhitekturi MVC kar poenostavi tako razvoj kot testiranje. MVC je arhitekturni model za implementacijo uporabniških vmesnikov [8]. Aplikacijo deli na tri medsebojno povezane dele ter s tem loči notranjo reprezentacijo informacij od tega, kako jo je program dobil od uporabnika ter kako je uporabniku predstavljena. Osnovna komponenta je model, ki vsebuje podatke, pravila, logiko in funkcije. View oziroma predstavitev je kakršen koli izpis informacij (recimo grafi, tabele, ...). Zadnji element je kontrolnik, ki poveže preostala dva dela. Kontrolnik sprejema vnose uporabnika ter jih pretvarja v ukaze za model in view, kakor je prikazano na sliki 3.3.

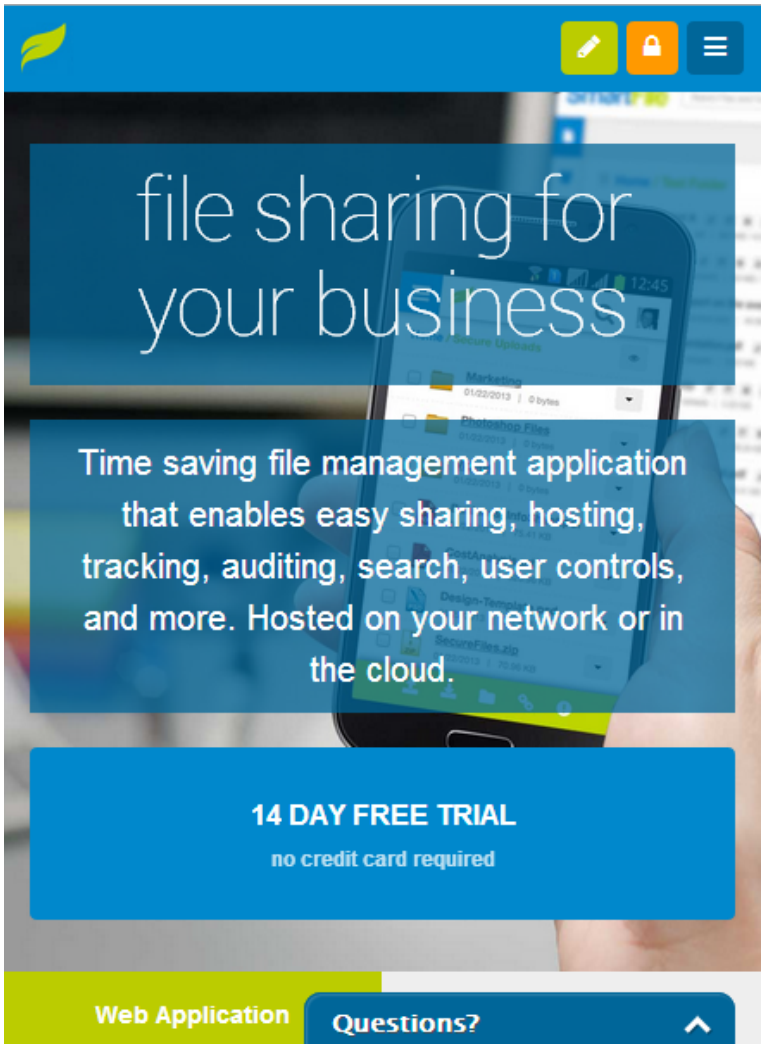


Slika 3.4: Izgled spletne strani narejene s Twitter Bootstrap v velikem oknu.

3.6 Twitter Bootstrap

Twitter Bootstrap je zbirka orodij, ki nam olajša izdelovanje spletnih aplikacij. Vsebuje oblikovne predloge, ki temeljijo na HTML in CSS, za tipografijo, obrazce, gumbe, navigacijo in ostale komponente za razvoj vmesnika kot tudi JavaScript razširitve [10]. Bootstrap podpira vse novejše verzije glavnih brskalnikov.

S pomočjo Twitter Bootstrapa lahko zelo enostavno zgradiš dinamično in odzivno spletno aplikacijo. Z uporabo njegovih predlog za različne komponente smo vgradili lepe in odzivne drsnike, gumbe in vnosna polja. Vsi elementi so zelo odzivni in uporabniku s subtilnimi spremembami sporočajo, kaj se dogaja, prav tako je zelo enostavna vgradnja sporočil o pravilnosti uporabnikovih vnosov. Twitter Bootstrap nam tudi olajša izgradnjo aplikacije, katere izgled se prilagaja velikosti uporabnikovega zaslona (ali brskalniškega okna). Ob spremembah velikosti okna se vsi elementi lepo porazdelijo ter spremenijo obliko tako, da še vedno ohranijo funkcionalnost, svoj izgled pa prilagodijo novim razmeram. Primer lahko vidite na slikah 3.4 in 3.5.



Slika 3.5: Izgled iste spletne strani iz slike 3.4 v malem oknu.

Poglavje 4

Theme Editor

Na projektu sem sodeloval prav na vseh področjih. Od zasnove arhitekture, zipanju paketov, spreminjanju klienta, a največ časa in dela sem namenil urejevalniku izgleda, ki smo ga poimenovali Theme Editor. Kot sem že omenil, nam sama zasnova omogoča dve vrsti prilagoditve uporabniškega vmesnika. V prvi vrsti definiramo delovanje in postavitev uporabniškega vmesnika, torej definiramo predlogo. Na tej predlogi pa potem z nastavljanjem barv, slik in animacij spreminjamo sam izgled določenih elementov, torej spreminjamo temo. Za vsako predlogo lahko naredimo poljubno število tem. Theme Editor je spletna aplikacija, ki nam omogoča kreiranje in urejanje tem.

Pred razvojem tega sistema to nista bili dve ločeni entiteti. Vsak vmesnik je bil hkrati predloga in tema, koda pa se je prepletala. Prva naloga je torej bila ločitev zapisa uporabniškega vmesnika na dva ločena dela, ki bi spreminjanje izgleda uporabniškega vmesnika iz dela programerja spremenila v administrativno delo. To omogoča hitrejše, enostavnejše in uporabnikom prijaznejše spreminjanje. Ponudnik IPTV ne rabi več zaprošati za spremembe, kar mora potem nekako priti v razvojni cikel, ter potem čakati na nov paket ter ga postaviti na sistem.

4.1 Definicija Predloge

Da lahko urejevalnik tem sploh deluje, potrebuje predlogo, nad katero potem dela spremembe. Predloga je definirana v notaciji JSON in opisuje vse vizualne elemente, ki jih lahko spreminjamo. Dejansko opišejo zbirko HTML oblikovnih parametrov, ki so nam na voljo za spreminjanje, hkrati pa že vsebujejo privzete vrednosti za te parametre. Poglejmo si primer take definicije.

Primer 4.1 *Majhen del kode, ki opisuje predlogo.*

```
{"meta": { "api-level": 1, "name": "themed", "mime": "application/themeb+json"},
"text-style": {
 "title": "Text styles",
 "group": "common",
 "changeable-properties": [
  { "name": "Color", "selector": ".svg", "type": "color", "value": "#ffffff" },
  { "name": "Weight", "selector": ".svg", "type": "font-weight", "value": "normal",
   "constraint": {"list-range": ["normal", "bold"]} },
  { "name": "Font", "selector": ".svg", "type": "font-family", "value": "Caius",
   "constraint": {"list-range": ["Caius", "Bitstream Vera Sans", "Tin Birdhouse"]} }
 ],
 "leafs": {
  "text-alternative": {
   "title": "Alternative",
   "changeable-properties": [
    { "name": "Color", "selector": ".text-alternative, .textAlternative,
     .menuItem.selected", "type": "color", "value": "#0099cc" },
    { "name": "Weight", "selector": ".text-alternative, .textAlternative",
     "type": "font-weight", "value": "normal", "constraint": {
      "list-range": ["normal", "bold"]}
    },
    { "name": "Font", "selector": ".text-alternative, .textAlternative",
     "type": "font-family", "value": "Caius", "constraint": {
      "list-range": ["Caius", "Bitstream Vera Sans", "Tin Birdhouse"]}
    }
   ]
  },
  "text-disabled": {
```

```
   "title": "Disabled UI text",
   "changeable-properties": [
    {"name": "Color", "selector": ".text-disabled", "type": "color", "value": "#828282"},
    {"name": "Weight", "selector": ".text-disabled", "type": "font-weight", "value": "normal",
     "constraint": {"list-range": ["normal", "bold"]}},
    {"name": "Font", "selector": ".text-disabled", "type": "font-family", "value": "Caius",
     "constraint": {"list-range": ["Caius", "Bitstream Vera Sans", "Tin Birdhouse"]}}
   ]
  },
  "text-contrasted": {
   "title": "Text on color inverted background",
   "changeable-properties": [
    {"name": "Color", "selector": ".text-contrasted", "type": "color", "value": "#0099cc"},
    {"name": "Weight", "selector": ".text-contrasted", "type": "font-weight",
     "value": "normal", "constraint": {"list-range": ["normal", "bold"]}},
    {"name": "Font", "selector": ".text-contrasted", "type": "font-family", "value": "Caius",
     "constraint": {"list-range": ["Caius", "Bitstream Vera Sans", "Tin Birdhouse"]}}
   ]
  },
  "text-selected": {
   "title": "Text on selected items",
   "changeable-properties": [
    {"name": "Color", "selector": ".selected", "type": "color", "value": "#05c9ee"},
    {"name": "Weight", "selector": ".selected", "type": "font-weight", "value": "normal",
     "constraint": {"list-range": ["normal", "bold"]}},
    {"name": "Font", "selector": ".selected", "type": "font-family", "value": "Caius",
     "constraint": {"list-range": ["Caius", "Bitstream Vera Sans", "Tin Birdhouse"]}}
   ]
  }
 }
}, ...
```

V primeru (4.1) je del kode, ki opisuje predlogo. Dokument se začne z oznako `meta` v katerem so zapisani nekateri ključni podatki kot so API verzija, za katero je bila predloga narejena ter ime predloge. `Title` vsebuje ime oziroma kratek opis lastnosti, za boljšo orientacijo v uredniku. `Group` je lahko nastavljen na `common` ali pa `screen`. `Common` pove, da gre za splošno

lastnost, ki se uporablja na večih mestih v grafičnem vmesniku, v tem primeru je to velikost in oblika teksta, `screen` pa nakazuje da gre za specifičen del vmesnika, na primer prijavno okno. Nadalje so opisani parametri tega elementa, ki jih lahko spreminjamo in njihove privzete vrednosti. V tem primeru definiramo stil teksta, kot če bi npr. napisali `text { ... }` v CSS datoteki. Nadalje imamo definirane alternativne nastavitve za tekst, katere se potem odražajo na alternativnih CSS razredih.

Obstaja nekaj pravil:

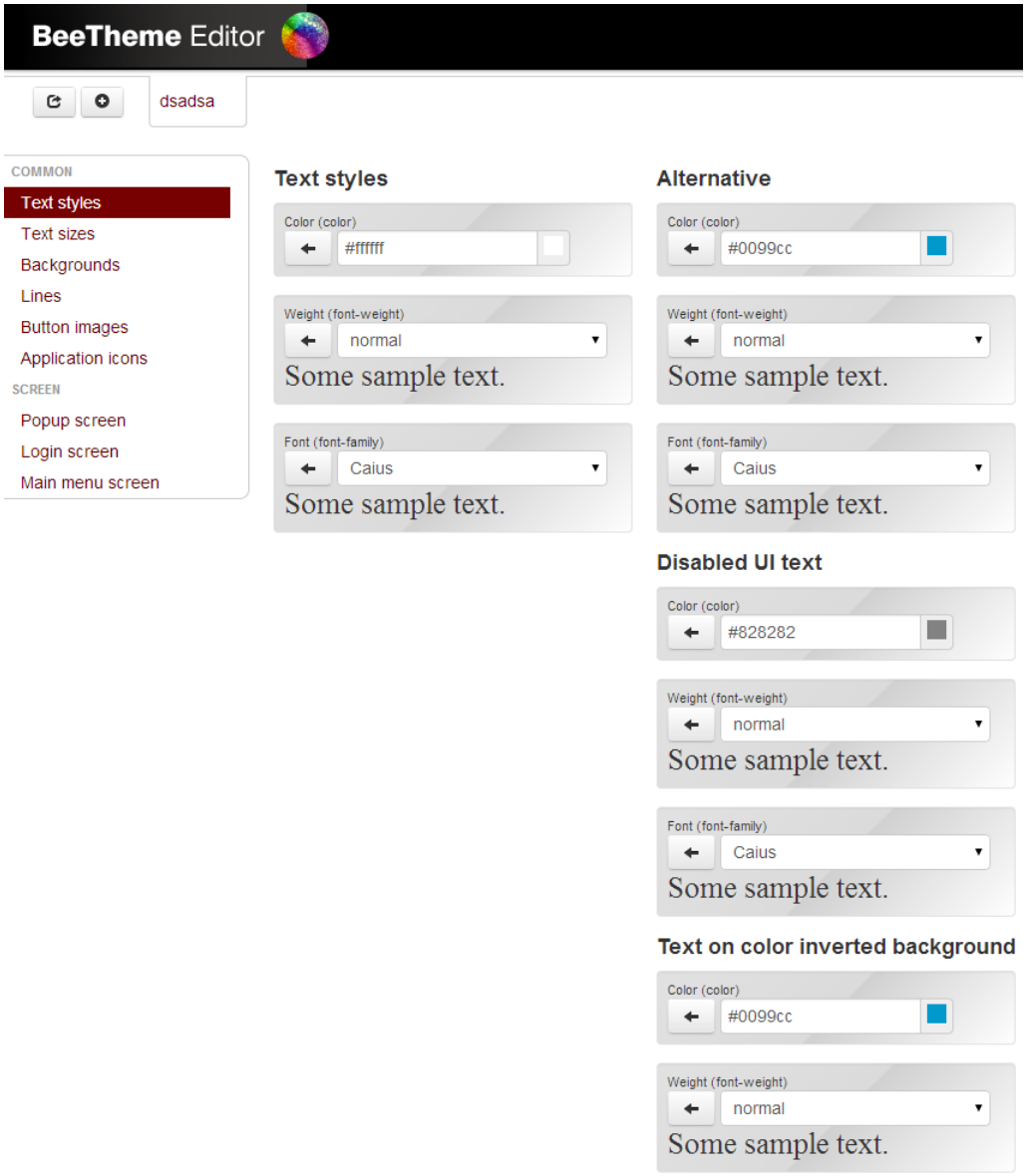
- Grafični elementi (elementi tipa `screen`) ne smejo imeti alternativnih `common` definicij.
- Spremenljive lastnosti izbirnika pri različnih elementih morajo biti unikatne znotraj področja delovanja.
- Definiran je samo en izgled, toda posamezni gradniki se lahko nahajajo kjerkoli znotraj paketa.
- Struktura posameznega elementa je lahko globoka največ dve stopnji.

4.2 Spletni Urejevalnik

Spletni urejevalnik je samostojna spletna aplikacija za urejanje tem. Na sliki 4.1 lahko vidimo, kako izgleda okno za urejanje predloge iz primera (4.1). Na levi strani so glavne lastnosti teksta, na desni pa alternative, ki so v predlogi napisane pod oznako `leaf`.

4.2.1 Uporaba

Ko odpremo urejevalnik se iz paketa naloži tema. Paket je arhiv datotek, ki so zapakirani v majhen paket WAR. V paketu so SVG datoteke uporabniškega vmesnika, vse potrebne slike, definicija teme in vse njene različice. Ko aplikacija prebere paket skonstruira kontrolne elemente za izbrano temo (kot je



Slika 4.1: Izgled urejevalnega okna za definicijo iz primera (4.1).

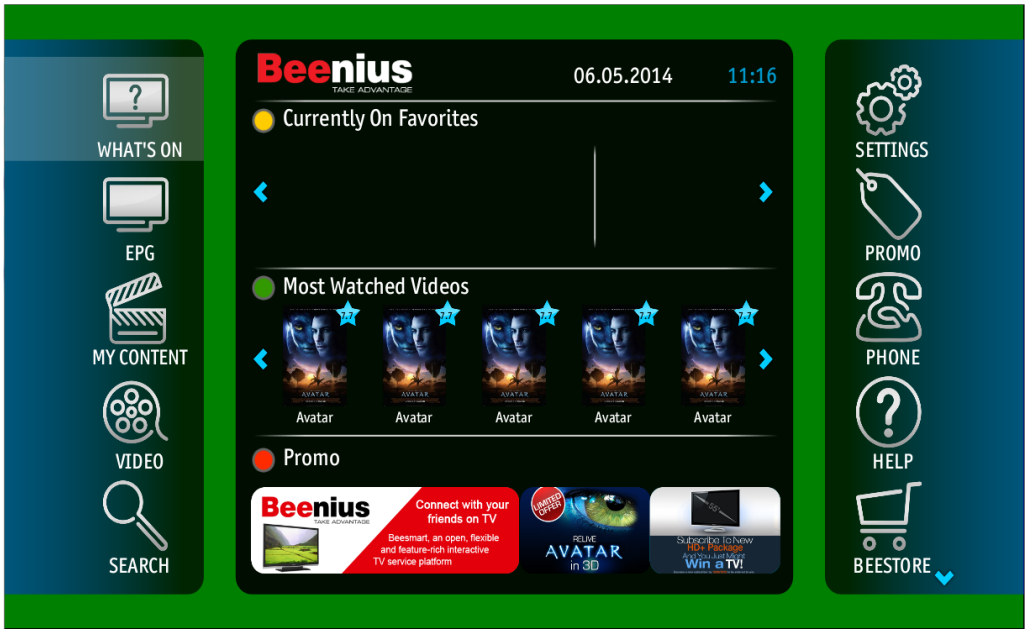
Slika 4.2: Enostaven prikaz IO dela urejevalnika izgleda klienta IPTV.

vidno na sliki 4.1). Na levi strani je menu v katerem izberemo kaj bi radi urejali, na desni pa so elementi, ki nam omogočajo spreminjanje videza.

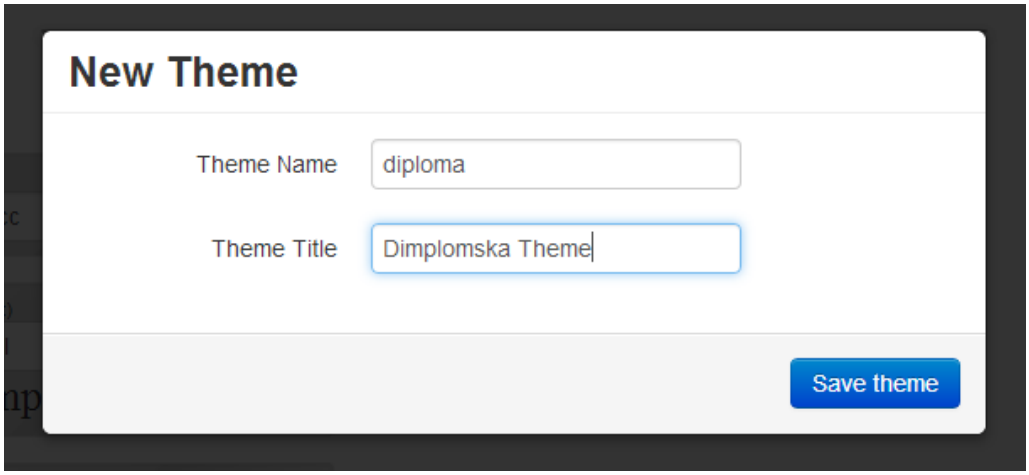
Nato lahko začnemo z urejanjem. Izberemo lahko že obstoječo variacijo teme ali pa začnemo z novo. Na sliki 4.3 lahko vidimo trenutni izgled klienta. V naslednjih korakih bomo spremenili njegov izgled. Če nimamo še nobene različice teme, bomo imeli v zgornem levem kotu zavihek z napisom untitled. Z dvojnim klikom se nam odpre okno, kjer lahko spremenimo ime novonastali različici teme 4.4.

Sedaj ko smo poimenovali temo bomo spremenili nekaj elementov. Če se spomnimo dela kode predloge iz primera (4.1) vemo, da je opisovala tekstovne lastnosti. Spremenimo torej barvo in tip pisave, kakor prikazuje slika 4.5, ponovno poženimo klienta, da se naložijo nove datoteke in poglejmo spremembe na sliki 4.6. Vidimo da so barve vseh tekstov spremenjene, prav tako pisava. Ura v desnem kotu uporablja alternativno nastavitev in je sedaj torej napisana z zeleno barvo v odebeljeni pisavi.

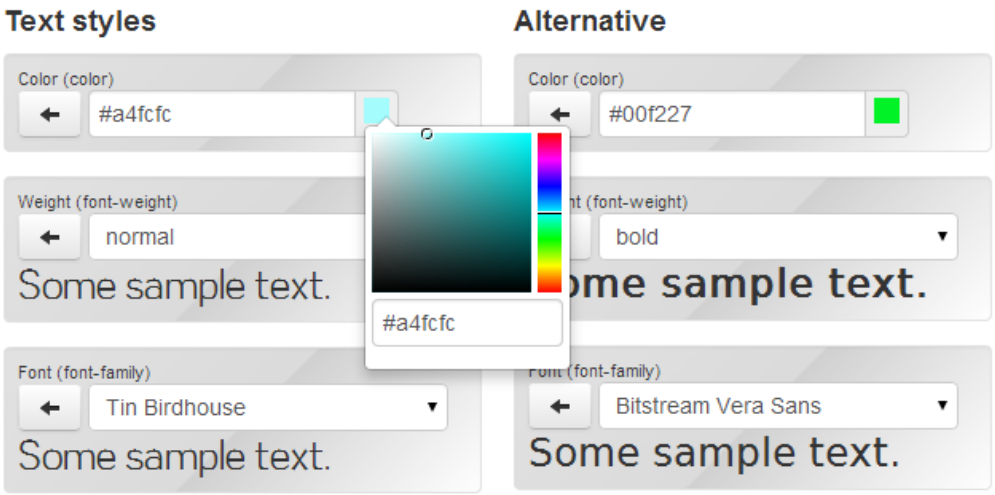
Poleg teh splošnih elementov ima glavni menu še svoj razdelek znotraj te predloge. To pomeni, da imamo v predlogi definiran element s skupino `screen` za glavni menu. V njej najdemo opis za ozadje glavnega menuja in če odpremo ta razdelek v Theme Editorju, lahko vidimo dve kontroli za določanje gradienta (slika 4.7). Rezultat vseh sprememb lahko vidite na sliki 4.8.



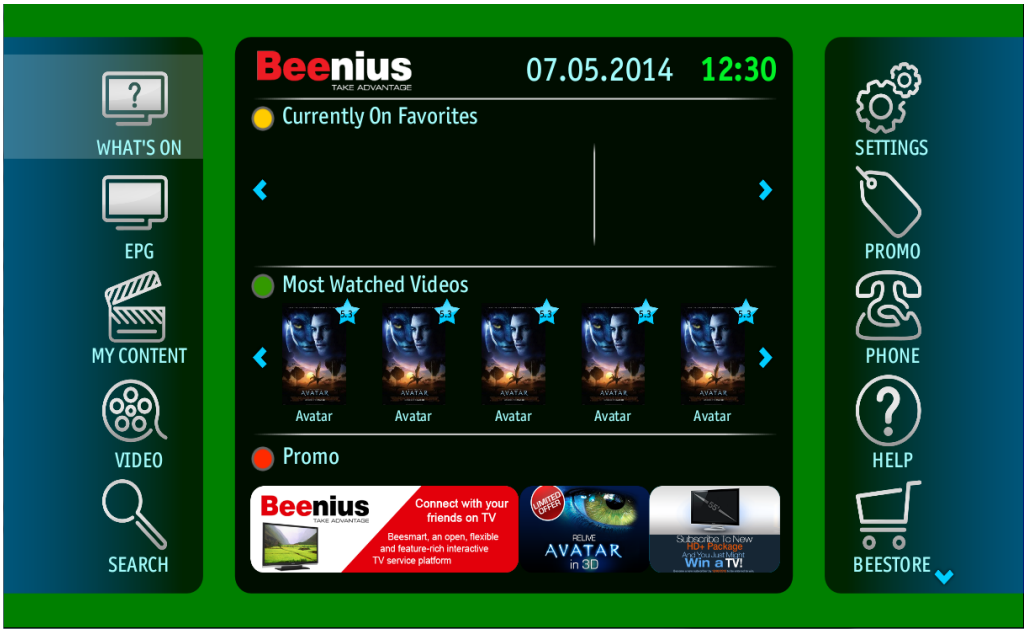
Slika 4.3: Izgled vmesnika z odprtim menujem. Na komunikatorju bi se namesto zelene barve v ozadju predvajal video.



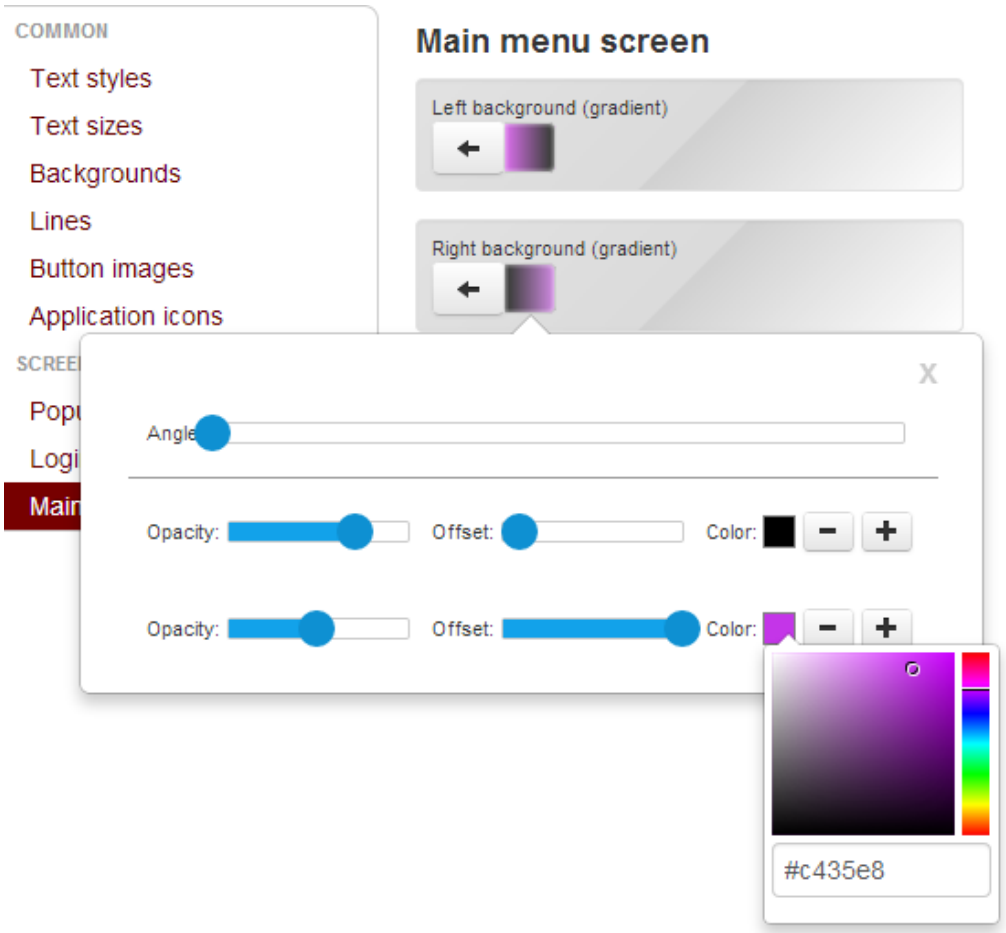
Slika 4.4: Okno za poimenovanje teme. Theme name je ime teme in predstavlja ime po katerem jo bo sistem poznal. Theme Title je namenjen za ljudem bolj prijazna imena/opise.



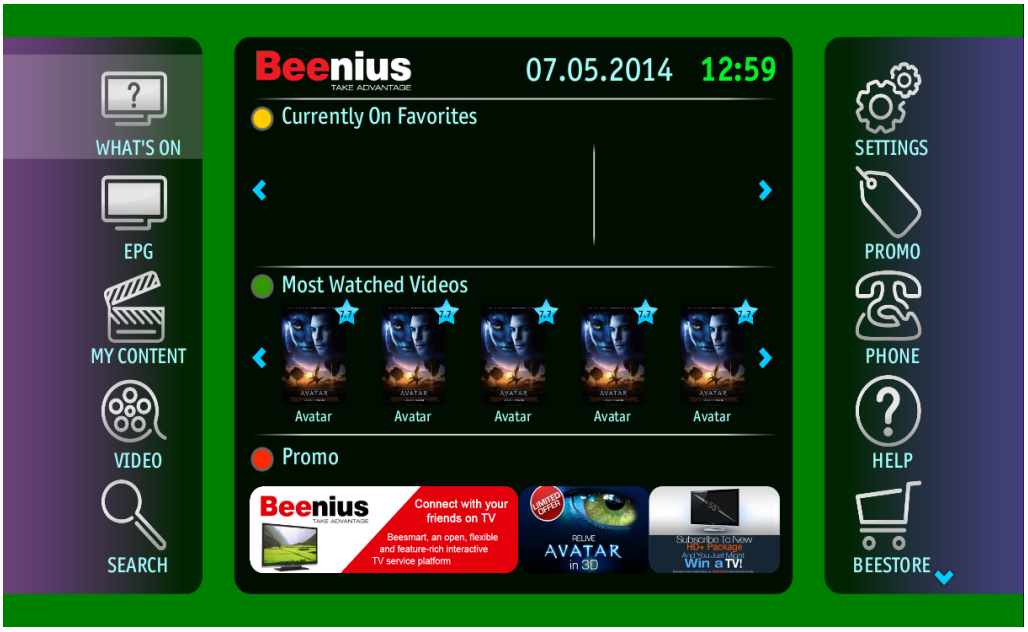
Slika 4.5: Spremenili smo barve, tip in debelino pisave.



Slika 4.6: Izgled uporabniškega vmesnika po spremembah na sliki 4.5.



Slika 4.7: Spreminjanje ozadja glavnega menuja. Na sliki vidimo tudi orodje za nastavljanje gradientov



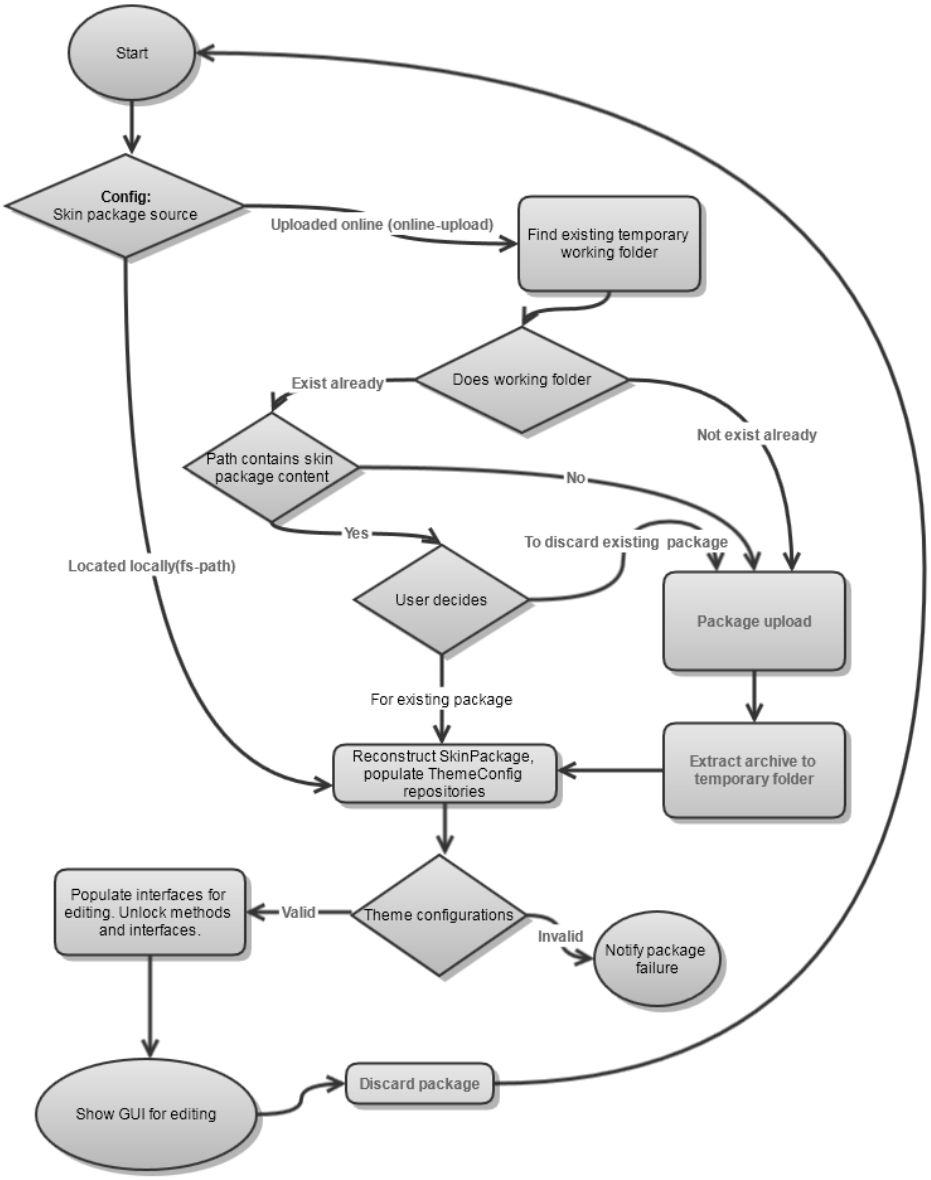
Slika 4.8: Končni rezultat vseh sprememb.

4.2.2 Delovanje

V prejšnjem poglavju smo pokazali uporabo urejevalnika na primeru, nismo pa povedali skoraj nič o procesih, ki se odvijajo v ozadju. V tem poglavju bom na istem primeru uporabe predstavil še to.

Najprej torej odpremo urejevalnik. V ozadju se poišče definicija teme 4.9. Definicija teme je, kot je prikazano v primeru (4.1), zapisana v formatu JSON in opisuje vse akcije, ki jih je administrator zmožen narediti. Ta definicija se razčleni (parsa) z regex izrazi. Z regex izrazi torej dobimo pomembne vrednosti iz dokumenta, hkrati pa predlogo tudi validiramo. Če so v predlogi napake, se na tem mestu izvajanje ustavi, uporabniku pa se skuša pomagati z informativnimi sporočili o napaki, za katere najpogostejše napake vsebujejo tudi predloge, kako problem rešiti.

Če je s predlogo vse v redu, se zgenerirajo kontrolni elementi. Če imamo že kako različico teme za to predlogo, iz dodatne datoteke JSON te deviacije pridobijo nastavitve in se kontrolni elementi urejevalnika nastavijo na te vre-



Slika 4.9: Proces inicializacije Theme Editorja.

dnosti, sicer se zgenerira nova različica s privzetimi vrednostmi iz predloge in privzetim imenom. Naredi se nova datoteka JSON, ki vsebuje privzete vrednosti, hkrati pa še datoteka SVG, ki ima te iste vrednosti definirane v brskalniku in komunikatorju prijaznem formatu SVG v obliki razredov CSS. Skopirajo se tudi vse privzete slike oziroma reference nanje.

Vsakič ko spremenimo parameter, se tema samodejno posodobi. Theme Editor v ozadju zgenerira novo verzijo SVG dokumenta, ki definira vse potrebne razrede CSS za oblikovanje vmesnika. Razredi so zgenerirani na podlagi opisa teme, vrednosti pa so vzete iz urejevalnika. Primer zgeneriranih razredov za našo temo je za datoteko SVG prikazan v primeru (4.2), generirano kodo JSON pa lahko vidite v primeru (4.3). Poleg razredov se generirajo tudi nekateri grafični elementi.

V prejšnjem poglavju smo poleg teksta spremenili tudi ozadje glavnega menuja. Spomnimo se, da je glavni menu v predlogi opisan kot `screen` in ima v urejevalniku lastno okno v katerem smo lahko nastavili dva gradienta, ki sta predstavljala ozadje menuja. Editor je za gradienta zgeneriral veljavno kodo SVG, ki je predstavljena v primeru (4.4) ter JSON, ki je na ogled v primeru (4.5).

Primer 4.2 *Razredi CSS iz datoteke SVG teme "diploma", ki smo jo naredili v primeru. Vrednosti se ujemajo z nastavitvami, ki smo jih nastavili na sliki 4.5.*

```
.svg{fill:#a4fcfc}
.svg{font-weight:normal}
.svg{font-family:Tin Birdhouse}
.text-alternative, .textAlternative, .menuItem.selected{fill:#00f227}
.text-alternative, .textAlternative{font-weight:bold}
.text-alternative, .textAlternative{font-family:Bitstream Vera Sans}
.text-disabled{fill:#828282}
.text-disabled{font-weight:normal}
.text-disabled{font-family:Caius}
```

Primer 4.3 *Datoteka JSON, ki se je zgenerirala, ko smo naredili našo variacijo teme, poimenovano "diploma". Vse vrednosti so aktualne.*

```
{
 "meta": {
  "api-level": 1,
  "name": "diploma",
  "title": "Dimplomska Theme",
  "mime": "application/beethemedeviation+json"
 },
 "text-style": {
  ".svg": {
   "color": "#a4fcfc",
   "font-weight": "normal",
   "font-family": "Tin Birdhouse"
  }
 },
 "text-alternative": {
  ".text-alternative, .textAlternative, .menuItem.selected": {
   "color": "#00f227"
  },
  ".text-alternative, .textAlternative": {
   "font-weight": "bold",
   "font-family": "Bitstream Vera Sans"
  }
 },
 "text-disabled": {
  ".text-disabled": {
   "color": "#828282",
   "font-weight": "normal",
   "font-family": "Caius"
  }
 },
 "text-contrasted": {
  ".text-contrasted": {
   "color": "#0099cc",
   "font-weight": "normal",
   "font-family": "Caius"
```

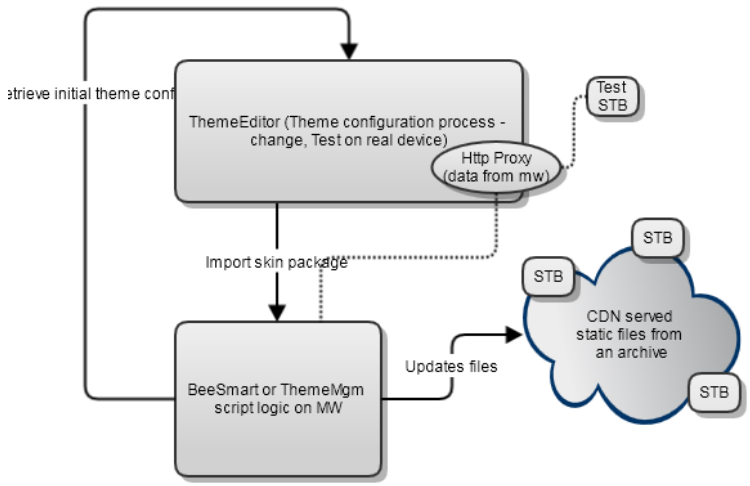
```
 }
},
"text-selected": {
 ".selected": {
  "color": "#05c9ee",
  "font-weight": "normal",
  "font-family": "Caius"
 }
},
"text-sizes": {
 ".svg": {
  "font-size": "23"
 },...
```

Primer 4.4 *SVG za ozadje glavnega menuja.*

```
<linearGradient id="screen-main-menu-left"
 spreadMethod="pad" gradientTransform="rotate(0)">
  <stop stop-color="#db01ff" offset="0%" stop-opacity="0.49"/>
  <stop stop-color="#000000" offset="90%" stop-opacity="0.71"/>
</linearGradient>
 <linearGradient id="screen-main-menu-right" spreadMethod="pad"
 gradientTransform="rotate(0)">
  <stop stop-color="#000000" offset="10%" stop-opacity="0.71"/>
  <stop stop-color="#8f00ff" offset="100%" stop-opacity="0.49"/>
</linearGradient>
```

Primer 4.5 *JSON za ozadje glavnega menuja.*

```
"screen-main-menu": {
 "#screen-main-menu-left": {
  "gradient": "linear(0, #db01ff7d 0%, #000000b5 90%)"
 },
 "#screen-main-menu-right": {
  "gradient": "linear(0, #000000b5 10%, #8f00ff7d 100%)"
 }
}
```



Slika 4.10: Proces osveževanja teme ob spremembah v Theme Editor.

Vse spremembe so torej že sprocesirane in shranjene v datotekah naše variacije poimenovane "diploma". Sedaj moramo le še ponovno zagnati naš komunikator in na zaslonu se bo izrisal nov zaslon. Logika hkrati tudi pregleda, če verzija middleware-a ustreza verziji GUI. Če je vse vredu, se zgenerira paket, ki se potem odloži na CDN 4.10.

Primer (3.2) in slika 3.1 sta predstavljata zelo enostaven primer, ki ne pokaže zmogljivosti vektorske grafike in formata SVG. Na sliki 3.2 lahko vidite nekoliko kompleksen primer, ki je dostopen tudi na internetu [6].

Poglavje 5

Dosedanji razvoj

V fazi načrtovanja smo projekt razdelili na dva dela. V prvem delu smo se odločili narediti urejevalnik predlog, v drugem delu pa bi omogočili popolno prilagodljivost delovanja klienta s pomočjo konfiguracijskih datotek.

Do sedaj smo naredili prvi del, torej urejevalnik predlog, kar zajema:

- specifikacija za GUI,
- spletni urejevalnik,
- sistem za dinamično serviranje predlog SVG,
- OpenAPI metode za deljenje konfiguracij predlog SVG,
- prototip predloge na novem sistemu.

Ta del je v veliki meri zaključen, nismo pa se še lotili drugega dela, ki bi projektu zagotovil veliko uporabnost in prilagodljivost.

Poglavje 6

Sklepne ugotovitve

V diplomskem delu smo naredili sistem za upravljanje izgleda klienta IPTV. Ta zajema prenovo klienta, nov strežnik za serviranje predlog, nov način pakiranja datotek in neodvisnost prikazovanja.

Z novim sistemom smo prevetrili delo pri prilagajanju izgleda klienta. Naredili smo močan urejevalnik tem, ki omogoča hitre spremembe videza in to moč prenese s programerjev na administratorje sistema. Ločili smo logiko od prestavitve ter omogočili večjo prilagodljivost uporabniškega vmesnika, ki sedaj omogoča tudi spremembe v obnašanju. Odstranili smo potrebo po ponovnih namestitvah ob spremembah uporabniškega vmesnika. Vgradili smo model MVC, ki je zmanjšal količino dela ob spremembah na middlewareu. V prihodnosti želimo podpreti še različne načine izrisovanja, kot je recimo HTML5, ter tako z isto aplikacijo podpreti tudi mobilne naprave ter celoten sistem združiti z že obstoječim administrativnim orodjem, saj sedaj teče kot samostojna aplikacija.

Sistem smo že predstavili nekaterim strankam in po sejmih, kjer so bili odzivi zelo pozitivni. Konec koncev gre za prvi realnočasovni urejevalnik izgleda za IP televizijo.

Literatura

[1] Jofferey E.F. Friedl, "Mastering Regular Expressions, Third Edition", O'Reilly Media Inc., 2006, str. 91-92

[2] Tom Huges-Croucher in Mike Wilson, "Node: Up and Running", O'Reilly Media Inc., 2012, str. 3-4

[3] Regular expression, dostopno 20.4.2015 na: http://en.wikipedia.org/wiki/Regular_expression.

[4] Regular expressions patterns, dostopno 21.4.2015 na: http://www.javascriptkit.com/javatutors/redev2.shtml

[5] Scalable Vector Graphics, dostopno 21.4.2015 na: http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[6] Svg example, dostopno 21.4.2015 na: http://jsfiddle.net/danielfilho/GdCcA/

[7] Grunt expression, dostopno 24.4.2015 na: http://gruntjs.com/

[8] Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra, "Head First Design Patterns", O'Reilly Media Inc., 2004, str. 529-531

[9] AngularJS, dostopno 24.4.2015 na: http://en.wikipedia.org/wiki/AngularJS

[10] Bootstrap (front-end framework), dostopno 24.4.2015 na:
http://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)