Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jan Sušnik

Razvoj večplatformnih aplikacij s pomočjo spletnih tehnologij za področje TV sporedov

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Mira Trebar

Ljubljana, 2017

Copyright. Rezultati diplomske naloge so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavo in koriščenje rezultatov diplomske naloge je potrebno pisno privoljenje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Spletne tehnologije omogočajo razvoj aplikacij za zelo različna področja. Kandidat naj v diplomskem delu predstavi njihovo uporabo z možnostjo implementacije na različnih platformah pri zasnovi in izdelavi spletne, namizne in mobilne aplikacije. Predstavljena rešitev naj vključuje ogrodja Angular, Electron in NativeScript za področje zajema podatkov in predstavitve televizijskih sporedov. Na osnovi dejanske namestitve in izvedenih testiranj naj preuči njihovo ustreznost ter poda njihove prednosti in slabosti.

Zahvaljujem se mentorici doc. dr. Miri Trebar za pomoč in nasvete pri izdelavi diplomske naloge.

Posebna zahvala gre staršem in sorodnikom, ki so me v času študija podpirali. Zahvaljujem se tudi prijateljem za motiviranje, sodelovanje in pomoč pri študijskih obveznostih.

Zahvaljujem se podjetju IECom d.o.o. in vsem posameznikom, ki so omogočili ali kakorkoli pripomogli k nastanku projekta in idej zajetih v diplomski nalogi.

Zahvala gre tudi vsem, ki so poskrbeli za usmerjanje v okviru izobraževanja ter na kakršenkoli način doprinesli k mojemu znanju.

Kazalo

Povzetek

Abstract

1 Uvod **1**

1.1 Motivacija 1

1.2 Cilji 2

1.3 Struktura diplomske naloge 2

2 Televizijski sporedi **3**

2.1 Priprava sporedov 5

2.2 Prikaz sporedov 11

3 Pregled ključnih tehnologij **19**

3.1 Spletne tehnologije in aplikacije 20

3.2 HTML, CSS in SASS 22

3.3 JavaScript in TypeScript 23

3.4 Enostranske aplikacije 24

3.5 RxJS 25

3.6 Angular 28

4 Spletna aplikacija **33**

4.1 Namen uporabe 33

4.2 Arhitektura in struktura aplikacije 35

4.3 Izbira dodatnih knjižnic 38

4.4 Nadzorni gradniki 38
4.5 Uporabnikova pot 46
4.6 Dodatni izzivi 59

5 Namizna aplikacija 71
5.1 Electron 71
5.2 Namen uporabe 73
5.3 Arhitektura in struktura aplikacije 74
5.4 Izbira dodatnih knjižnic 78
5.5 Uporabnikova pot 80
5.6 Dodatni izzivi 88

6 Mobilna aplikacija 95
6.1 NativeScript 96
6.2 Namen uporabe 98
6.3 Arhitektura in struktura aplikacije 98
6.4 Izbira dodatnih knjižnic 103
6.5 Uporabnikova pot 104
6.6 Dodatni izzivi 112

7 Zaključek 131
7.1 Sklepne ugotovitve 132
7.2 Spremna misel 134

Literatura 135

Seznam uporabljenih kratic

kratica	angleško	slovensko
AJAX	asynchronous JavaScript and XML	asinhroni JavaScript in XML
AOT	ahead of time	predčasno prevajanje
API	application programming interface	aplikacijski programski vmesnik
APK	Android package kit	paket z orodji za Android
CSS	cascading style sheets	kaskadne oblikovne predloge
DOM	document object model	objektni model dokumenta
EPG	electronic program guide	elektronski programski vodič
FTP	file transfer protocol	protokol za prenos datotek
HTML	hypertext markup language	jezik za označevanje nadbesedila
HTTP	hypertext transfer protocol	protokol za prenos hiperteksta
IP	internet protocol	internetni protokol
IPC	interprocess communication	medprocesna komunikacija
IPG	interactive program guide	interaktivni programski vodič
ITV	interactive television	interaktivna televizija
JIT	just in time	prevajanje v času izvajanja
JSON	JavaScript object notation	zapis objekta JavaScript
JWT	JSON web token	spletni žeton JSON
ORM	object-relational mapping	objektno-relacijska preslikava

PWA	progressive web app	napredna spletna aplikacija
REST	representational state transfer	predstavitveni prenos stanja
SASS	syntactically awesome style sheets	oblikovne predloge z nadgrajeno sintakso
SDK	software development kit	orodja za razvoj programske opreme
SOAP	simple object access protocol	protokol za enostaven dostop do objekta
SPA	single-page application	enostranska aplikacija
SQL	structured query language	strukturiran povpraševalni jezik
STB	set-top box	digitalni sprejemnik
TV	television	televizija
XML	extensible markup language	razširljivi označevalni jezik

Povzetek

Naslov: Razvoj večplatformnih aplikacij s pomočjo spletnih tehnologij za področje TV sporedov

Avtor: Jan Sušnik

Spletne tehnologije predstavljajo eno izmed možnosti za razvoj različnih vrst aplikacij, ki postaja vse bolj priljubljena. Diplomska naloga vključuje razvoj spletne, namizne in mobilne aplikacije, ki smo jih izdelali s pomočjo ogrodij Angular, Electron in NativeScript. Slednja omogočajo pripravo aplikacij za več končnih platform. Osrednje področje so zajemali televizijski sporedi, katerih procese smo podrobno predstavili. Opisali smo akterje, vključene v postopek nastajanja in posredovanja sporedov ter njihovo glavno vlogo. Za podporo vsakemu procesu smo izdelali pripadajočo aplikacijo, ki uporabniku ponuja intuitiven vmesnik in omogoča učinkovito uporabo. Osredotočili smo se na predstavitev namena aplikacije, njene arhitekture, primere uporabe in dodatne izzive, s katerimi smo se srečali tekom razvoja. Preučili smo ustreznost aplikacij za uporabo v produkcijskem okolju. Na osnovi izdelanih rešitev smo izpostavili prednosti in slabosti uporabljenih ogrodij.

Ključne besede: večplatformne aplikacije, spletne tehnologije, Angular, Electron, NativeScript, TV sporedi.

Abstract

Title: Development of cross-platform applications with web technologies for the field of TV listings

Author: Jan Sušnik

Web technologies represent one among the choices for development of different types of applications, which is becoming increasingly popular. In the diploma thesis we discussed development of web, desktop and mobile applications, created using Angular, Electron and NativeScript frameworks. They support preparation of cross-platform applications. The core field was covered by television listings, so we presented each of its processes in detail. We described actors involved in the process of creating and forwarding listings alongside their main role. In order to support each process, we developed a corresponding application that offers an intuitive interface to the user and enables efficient use. We focused on the presentation of application's intent, its architecture, use cases and additional challenges that we encountered during development. We examined the relevance of applications for use in production environment. Based on developed solutions, we highlighted the strengths and weaknesses of used frameworks.

Keywords: cross-platform applications, web technologies, Angular, Electron, NativeScript, TV listings.

Poglavje 1

Uvod

1.1 Motivacija

Spletne tehnologije so dandanes na področju računalništva nepogrešljive. V številnih aplikacijah predstavljajo glavni člen komunikacije med računalniki, kot tudi komunikacije med računalnikom in človekom. V svojih začetkih so bile uporabljene večinoma za prikaz informacij, z razvojem pa so vse bolj postajale namenjene interaktivnosti z uporabnikom. Napredek na področju spletnih tehnologij se iz dneva v dan povečuje, kar potrjuje tudi dejstvo, da se ostale vrste aplikacij selijo na splet. Na nasprotni strani odjemalskih spletnih aplikacij pa najdemo mobilne aplikacije, ki običajno za svoje delovanje uporabljajo storitve zgrajene na osnovi spletnih tehnologij. Trend aplikacij torej trenutno kaže v prid spletnim in mobilnim aplikacijam, navkljub temu pa se namiznim aplikacijam ni mogoče izogniti.

Zaradi širokega nabora programskih jezikov in ostalih tehnologij, ki jih je možno uporabiti pri razvoju vseh vrst aplikacij pa nastane problem. Za pristno delovanje posameznih aplikacij je potrebno dobro poznavanje končnih platform. V zadnjih nekaj letih se prav s tem problemom soočajo na področju spletnih tehnologij, saj poskušajo nadomestiti razvoj domorodnih (ang. native) aplikacij prav z uporabo tehnologij, ki jih uporabljamo za izdelavo spletnih aplikacij.

1.2 Cilji

Cilj diplomske naloge je predstaviti možnosti razvoja različnih vrst aplikacij s pomočjo spletnih tehnologij. V okviru diplomske naloge bodo razvite tri vrste aplikacij – spletna, namizna in mobilna. Za realizacijo teh bomo uporabili področje televizijskih sporedov, katerega procese lahko pokrijemo z aplikacijami, ki vključujejo različne vidike uporabe.

Na osnovi razvoja vseh treh aplikacij želimo ugotoviti, če se predstavljene možnosti obnesejo dovolj dobro za razvoj produkcijskih aplikacij in ne zgolj prototipov. Zanima nas kakšne so prednosti in slabosti pri razvoju in razvojnih tehnologijah vsake aplikacije.

1.3 Struktura diplomske naloge

V drugem poglavju je predstavljeno področje televizijskih sporedov, ki zajema osrednjo temo za razumevanje nadaljnjih poglavij. Opisuje osnovne pojme sporedov in glavna procesa obravnave njihovih podatkov.

Tretje poglavje je namenjeno pregledu ključnih tehnologij, uporabljenih v okviru razvitih aplikacij. Na začetku je na kratko predstavljena zgodovina spletnih tehnologij in trenutni trendi, nakar sledi opis vsake uporabljene tehnologije.

V četrtem, petem in šestem poglavju so za vsako izdelano aplikacijo najprej izpostavljene morebitne dodatne uporabljene tehnologije. Poglavja se osredotočajo predvsem na namen uporabe posamezne aplikacije, njeno arhitekturo in primere uporabe. Na koncu so izpostavljeni dodatni izzivi, ki jih je mogoče pričakovati pri razvoju tovrstnih aplikacij.

V zadnjem, sedmem poglavju so povzeti prispevki diplomske naloge in predstavljene ugotovitve, dognane na osnovi prejšnjih poglavij.

Poglavje 2

Televizijski sporedi

Televizija je uveljavljen elektronski medij, ki je namenjen komunikaciji med ponudnikom informacij in občinstvom. Struktura po kateri deluje je v vsakdanjem življenju nekaj povsem običajnega. Kljub temu bomo opisali nekaj najpomembnejših pojmov, ki se pojavljajo v televizijskem svetu, saj predstavljajo osnovo za nadaljnje razumevanje poglavja.

Televizijski spored ali krajše TV spored definiramo kot zaporedje dogodkov v določenem časovnem obdobju. Vsak dogodek ima določen vsaj datum in čas začetka ter trajanje, v okviru katerega se predvaja neka vsebina. Ta običajno obravnava točno določeno tematiko, ki jo gledalec prepozna na podlagi njenega naslova. V kolikor so znani vsi trije omenjeni podatki, tak dogodek lahko označimo kot televizijska oddaja.

Televizijsko oddajo ali samo oddajo definiramo kot medijsko vsebino, ki lahko predstavlja resničen, splošen ali izmišljen pogled na določeno stvar. Na podlagi tega oddaji določimo tip, s katerim jo gledalec identificira. Lahko gre za film ali nadaljevanko, športni dogodek, splošni ali propagandni program oziroma je oddaja le nadomestilo za čas, ko na programu ni vsebine.

Televizijski kanal ali program predstavlja medij, za katerega se pro-

ducira oddaje, ki jih uvrstimo na TV spored. Navadno se na posameznem programu predvajajo oddaje istega področja, z izjemo nacionalnih in nekaterih komercialnih programov. Za uvrstitev oddaj v spored skrbi programski direktor ali urednik programa, ki deluje znotraj televizijske hiše.

Programski dan ali dan sporeda je časovno obdobje, ki vsebuje zaporedje oddaj za določen datum in se ponavadi prične ob 6. uri zjutraj. Za nekatere kanale ta omejitev ne obstaja, saj vsebino predvajajo neprekinjeno, medtem ko jo drugi strogo upoštevajo in v nočnem času predvajajo infokanale ali ne oddajajo.

Televizijska hiša je ustanova, ki ima v lasti enega ali več TV kanalov. Pojem se v zadnjem času nadomešča z medijsko hišo, saj vsebine niso več dostopne le preko televizijskih sprejemnikov, temveč tudi na drugih medijih, kot je internet. Večje TV hiše, ki delujejo mednarodno, priprave sporedov ne vodijo same, temveč jih za njih pripravljajo regijski ali lokalni distributerji.

Ponudnik televizijskih sporedov je organizacija, ki se ukvarja z združevanjem TV sporedov iz izvornih virov za večjo množico kanalov. Skrbi za točnost, pravilnost in konsistentnost podatkov ter poskuša zagotoviti spored za najdaljše možno obdobje. Sporede lahko obogati z dodatnimi podatki, od katerih ima uporabnik storitev dodatne koristi.

Ponudnik televizijskih storitev je podjetje, ki omogoča prenos televizijskega pretoka od njenega lastnika (pravic) do gledalca. Ponudnik ima za nadaljnje predvajanje pretoka omejene pravice, zato obstaja možnost, da nekaterih oddaj ne sme predvajati. Prav tako mora poskrbeti za omejitev ogleda na podlagi naročene sheme in dovoljenih naprav.

V nadaljevanju bomo spoznali proces prenosa sporeda od njegovega avtorja do gledalca TV kanala, ki je značilen za IP (ang. Internet Protocol) in

kabelsko televizijo ter nekatere druge vrste medijev.

2.1 Priprava sporedov

V procesu priprave sporedov sodelujeta dva glavna akterja – televizijska hiša ali njen distributer (v nadaljevanju vir podatkov) in ponudnik televizijskih sporedov. Omejili se bomo le na vidik priprave sporedov s strani slednjega. Proces sestoji iz treh korakov, ki obravnavajo zajem vhodnih podatkov, obdelavo in pripravo izhoda. V rezultat procesa je vključen tudi uporabnik, ki sporede na koncu uporabi kot del dejavnosti ali storitve, ki jo ponuja.

2.1.1 Zajem vhodnih podatkov

Ponudnik TV sporedov za vsak program potrebuje izvorni dokument, ki ga zagotavlja vir podatkov in jamči za njegovo pravilnost. V njem je podan spored za določeno časovno obdobje, razdeljen na dogodke. Dokument predstavlja vhod na podlagi katerega ponudnik sporedov lahko izvaja nadaljnje akcije in je ključno sredstvo prvega koraka. Za obdelavo podatkov morajo dogodki ustrezati definiciji sporeda in oddaje, kar pomeni da ima posamezen dogodek določen vsaj začetek, trajanje in naslov.

Izvorni dokument je običajno (delno) strukturirana besedilna datoteka, ki jo vir podatkov pridobi iz informacijskega sistema z neposredno povezavo s sistemom za oddajanje programa – primeri takšnih datotek so HTML (ang. Hypertext Markup Language), XML (ang. Extensible Markup Language) in JSON (ang. JavaScript Object Notation). Pogosto je dokument tudi v obliki najbolj uporabljenih lastniških formatov za dokumente in preglednice. Večinoma so dokumenti zapisani v obliki, ki je med različnimi časovnimi obdobji nespremenjena. Obstajajo tudi povsem nasprotni primeri, kjer se oblika vsakokrat spremeni. Dokumenti različnih TV kanalov imajo redko enako obliko, običajno le v primerih, ko za njimi stoji isti vir podatkov. Posledično to pomeni, da splošno uveljavljene oblike za izmenjavo podatkov

ni. Kljub temu obstajajo viri podatkov, ki zagotavljajo dokumente v formatu XMLTV, znanemu kot *de facto* standard za izmenjavo TV sporedov. Primer takšnega dokumenta je na sliki 2.1.

```
<programme start="20171008022000 +02:00" stop="20171008023000 +02:00" channel="SLO1">
  <title><![CDATA[Vreme]]></title>
  <title lang="sl"><![CDATA[Vreme]]></title>
  <sub-title lang="sl"><![CDATA[Vreme]]></sub-title>
  <desc lang="sl"><![CDATA[Vreme je na sporedu vsak dan po Poročilih, pred in po Dnevniku
  ter po Odmevih.]]></desc>
  <category lang="sl"><![CDATA[Informativni program]]></category>
  <category lang="sl"><![CDATA[Vreme]]></category>
  <language lang="sl"><![CDATA[Slovenski]]></language>
  <country lang="sl"><![CDATA[Slovenija]]></country>
</programme>
```

Slika 2.1: Zapis podatkov oddaje v formatu XMLTV.

Dodatno vir podatkov lahko zagotovi še medijske datoteke na katere se sklicuje v izvornem dokumentu ali jih le priloži. Tovrstne datoteke so ločene od izvornih dokumentov in se jih v procesu priprave sporedov smatra kot dodatno sredstvo. Medijske datoteke so povečini slike prizorov iz oddaj, plakati ali logotipi oddaj. Vse bolj se uveljavlja tudi video vsebina, kot so napovedniki.

Način izmenjave datotek med virom podatkov in ponudnikom TV sporedov je stvar njunega dogovora. Najpogosteje se za izmenjavo uporablja spletne strani in spletne storitve, e-pošto, FTP (ang. File Transfer Protocol) in datotečno shrambo v oblaku.

Prvi korak obravnava prenos datotek od vira podatkov k ponudniku sporedov, preverjanje ustreznosti izvornega dokumenta in zajem oddaj iz dokumenta. Rezultat koraka je vsaj dokument, ki zadostuje pogojem za nadaljnjo obdelavo, saj v nasprotnem primeru ni mogoče pripraviti pravilnega sporeda. Primer: Nacionalna TV hiša (vir podatkov) za sporede svojih programov nudi javno dostopen API (ang. Application Programming Interface), do katerega dostopamo na podlagi TV kanala in datuma, za katerega želimo prejeti spored. Kot odgovor na podan zahtevek dobimo strukturirano besedilno datoteko v obliki XML (izvorni dokument), ki vsebuje zaporedje oddaj

le za izbrani datum. Vsaka oddaja ima označen začetek in konec predvajanja, naslov v več jezikih in dodatne vsebinske podatke. Tak dokument je primeren za nadaljnjo obdelavo, ki sledi zatem, ko je spored za izbran datum objavljen ali spremenjen.

2.1.2 Obdelava podatkov

V korak obdelave štejemo zapis podatkov v lasten sistem, preverjanje točnosti, pravilnosti in konsistentosti oddaj ter bogatenje osnovnih podatkov z dodatnimi. Obdelavo podatkov lahko ponudnik sporedov izvrši na več načinov:

- **strojno** – vhodne podatke obdela računalnik s pomočjo algoritmov
- **človeško** – vhodne podatke ročno prepiše in uredi človek
- **kombinirano** – del koraka se opravi strojno, preostanek pa zaradi pravilnosti podatkov opravi človek

Način obdelave je povsem odvisen od kvalitete vhodnega podatka – izvornega dokumenta iz prvega koraka. V kolikor so podatki smiselno oblikovani jih je mogoče obdelati strojno, v nasprotnem primeru jih obdela človek. Pri izbiri načina obdelave se je potrebno osredotočiti na število novih oddaj na sporedu, pogostost osveževanja sporeda s strani vira podatkov, vključevanje dodatnih sredstev in ostale podrobnosti. V tem koraku pomembno vlogo igra predvsem sistem, ki omogoča urejanje sporedov. Ta je lahko povsem enostaven in le shranjuje zapise ter skrbi za pravilnost. Lahko pa poleg tega implementira še kompleksno logiko, ki skrbi za povezovanje oddaj z obstoječimi v sistemu. Na drugi strani namesto sistema to nalogo lahko opravlja človek. Navkljub vsem algoritmom, ki jih lahko implementiramo v sistem, je na koncu za najboljši rezultat pogosto potreben kombiniran način obdelave, sploh če želimo zagotoviti točnost podatkov v sporedu. Podatki v izvornih dokumentih so dvoumni, kar lahko avtomatiziramo le v primeru, da

ta vključuje dodatne podatke, ki pomagajo prepoznati oddajo. Za primer lahko vzamemo oddajo z naslovom *Novice*, kjer ima beseda različen pomen v slovenščini in angleščini. Brez dodatnih podatkov in baze znanja ni mogoče ugotoviti ali gre za informativno oddajo ali film.

Neodvisno od načina obdelave in zapisa podatkov v sistem je eden izmed pomembnih vidikov tudi izgled sporedov. Potrebno je zagotoviti:

- **točnost** – spored se ujema z izvornim dokumentom in je karseda ažuren glede na popravke s strani vira podatkov
- **pravilnost** – oddaje morajo biti v sporedu zapisane nedvoumno, tako da je gledalec lahko prepričan, da bo gledal želeno vsebino
- **konsistentnost** – oddaje morajo slediti ena drugi brez prekrivanj oziroma dve oddaji hkrati ne moreta biti predvajani

Dodatno pravilo v tem koraku je tudi filtriranje oddaj, ki so primerne za objavo v sporedu. Izvorni dokument namreč lahko vsebuje tudi oddaje kot so uvodne špice, oglasi in druge krajše vsebine, ki gledalca ne zanimajo neposredno.

Pod obdelavo podatkov štejemo tudi bogatenje z dodatnimi podatki, ki lahko vplivajo na gledalčevo odločitev o tem ali bo oddajo spremljal. Potreba po njih se je pojavila kot odziv napredka na področju pametnih naprav in povezovanja podatkov za predloge (oddaje s podobno vsebino, ki se prikažejo glede na pretekle oglede). Podatke razdelimo na osnovne in dodatne. Osnovni so tisti, ki oddajo poskušajo opisati z najmanj vsebine, dodatni pa razkrivajo celovitejši pogled nanjo in dopolnjujejo osnovne. Kateri podatki so vključeni kam je odvisno od ponudnika sporedov in njegovega principa dela. Primeri osnovnih podatkov so izvorni naslov, lokaliziran naslov in jezik oddaje. Primeri dodatnih podatkov so kategorija, žanr, opis, slike in video vsebina. Nekatere izmed omenjenih podatkov je možno pridobiti od obstoječih zunanjih virov, ki preko svojih storitev javno omogočajo urejanje, ocenjevanje in razpravo o oddajah.

Drugi korak obravnava obdelavo podatkov, ki za delovanje uporablja izvorni dokument. Tega ponudnik sporedov obdela s strojnimi ali človeškimi viri tako, da pripravi čimbolj pregleden in pravilen spored. Rezultat je spored, ki gledalcu za nek TV kanal da dober vpogled v njegovo vsebino in jo poskuša predstaviti čimbolj nedvoumno. Primer: Odločimo se za kombiniran način obdelave sporeda. Podatke iz izvornega dokumenta nacionalne televizije s pomočjo strojnega načina naložimo v sistem. Izvorni dokument poleg osnovnih podatkov vključuje tudi nekaj dodatnih. Vsako oddajo nato ročno pregledamo – preverimo ali so to podatki, ki jih je zagotovil vir podatkov in ali so pravilni ter konsistentni. Pri eni izmed oddaj ugotovimo, da je v naslovu oddaje napaka in jo popravimo. Nato najdemo, da pri oddaji opis ne ustreza dejanski oddaji na sporedu in ga popravimo. Nazadnje ugotovimo še, da je med koncem in začetkom dveh oddaj 5 minut premora in eno izmed oddaj podaljšamo. Celoten postopek tako ponavljamo dokler spored za neko obdobje ni v celoti urejen.

2.1.3 Priprava izhoda

Ko so podatki pripravljeni za objavo, jih je potrebno pripraviti v ustrezno obliko izhoda. Ta del je zelo podoben prvemu koraku, saj ponudnik sporedov pripravi datoteke s podatki in jih posreduje naprej v nadaljnjo obdelavo. Prav tako jamči, da so izhodni podatki pravilni in ažurni. Za razliko od prvega koraka mora ponudnik poskrbeti, da so podatki ustrezno oblikovani in pripravljeni za prikaz v končnih medijih. Ta korak lahko določa kaj vse mora biti obdelano v predhodnem, saj spored ne ustreza rezultatu, če nima podatkov, ki morajo biti vsebovani v izhodu tega koraka. Ko je izhod za neko obdobje uspešno pripravljen, je posredovan k uporabniku.

Ponudnik TV sporedov mora za uporabnika pripraviti sporede za vse kanale, za katere sta dogovorjena. Vsak kanal ima določen nabor podatkov, ki jih je mogoče pridobiti iz izvornega dokumenta ali zunanjih virov. Najmanjši obseg podatkov, ki se lahko pripravijo za izhod, so obvezni podatki oddaje. Seznam, ki določa kateri podatki bodo na izhodu poleg obveznih,

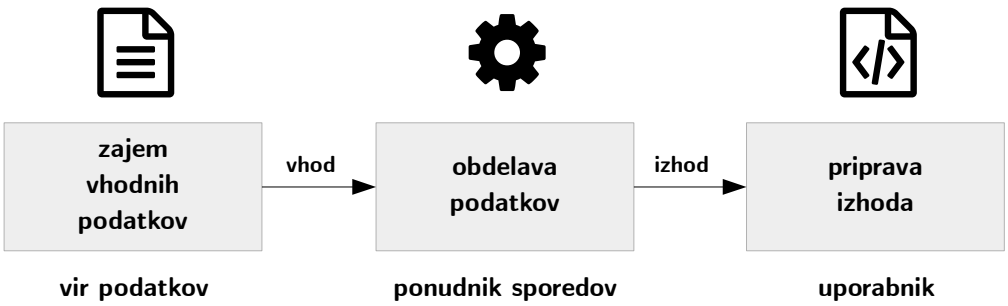
imenujemo shema. To običajno določi uporabnik na osnovi priljubljenosti kanalov in podatkov, ki so lahko del izhoda. Shema ponudniku TV sporedov med drugim pove, koliko truda mora vložiti v posamezen kanal, kar vpliva na korak obdelave podatkov. Poleg sheme se za vsak kanal določi še minimalno obdobje za katerega je spored dosegljiv. To je odvisno tudi od vira podatka – lahko je to nekaj dni vnaprej, lahko tudi več tednov. V nekaterih primerih se spremembe sporeda naredijo tudi za nazaj, predvsem kadar uporabnik omogoča ogled preteklih oddaj na kanalu.

Izhodne datoteke, ki vsebujejo spored morajo biti zapisane v strukturirani obliki. Razlog za to je v količini podatkov, ki so med seboj povezani, saj je potrebno poskrbeti, da ustrezajo pravi oddaji. Poleg tega je potrebno upoštevati, da uporabnik v svojem sistemu opravi le vnos podatkov, ki se mu jih zagotovi in ne izvaja celotne obdelave, kot je opisana v drugem koraku. Posledično je smiselno uporabiti splošno uporabljen standard oblike datotek, kot je XMLTV, ali pa v primeru da ta ne ustreza obsegu zagotovljenih podatkov izdelamo lastniškega. V tem primeru je dobro razmisliti o izdelavi sheme za preverjanje pravilnosti, ki jo pošljemo uporabniku, da po prenosu datoteke lahko preveri, če je med tem slučajno prišlo do napake. Prav tako je oblika datotek stvar dogovora med ponudnikom sporedov in uporabnikom. To velja tudi za način izmenjave, ki lahko temelji na primer na spletnih storitvah ali datotečnih sistemih v oblaku.

Tretji korak poskrbi za pripravo ustreznih podatkov v izhodno datoteko, ki ima strukturirano obliko. Rezultat je pravilno pripravljena izhodna datoteka, ki je uspešno posredovana uporabniku. Primer: Spored za naslednji programski dan smo uredili s popravki iz izvornega dokumenta in zajema vse podatke, ki so vključeni v shemo. Posledično je pripravljen na objavo, ki jo sproži neka akcija v sistemu. Na podlagi sheme se izdela strukturirana datoteka XML v lastniški obliki. Uporabnik nam omogoča, da mu datoteko pošljemo preko protokola SOAP (ang. Simple Object Access Protocol) in pred potrdivijo uspešnega prejema preveri, če se datoteka sklada z že prej izmenjano shemo. V primeru napake se mora ponoviti najmanj ta korak, v

primeru neustreznosti podatkov glede na shemo, pa pred njim še predhodni korak. S potrditvijo uporabnika o uspešnem prejemu se celoten proces priprave sporedov za programski dan zaključi.

Skozi opisan proces, katerega povzetek je predstavljen na sliki 2.2, smo spoznali kako poteka sprejem, obdelava in distribucija sporedov. Vsakega izmed korakov smo opisali na najbolj splošen način, saj zagotovo obstajajo razlike v podrobnostih med ponudniki sporedov. Proces prav tako predstavlja razloge, ki osmislijo odločitev uporabnika za uporabo vmesnega člena med virom podatkov in njim. To se izkaže predvsem takrat, ko mora ta zagotavljati sporede za večjo količino kanalov ali pa želi bolj kakovostno vsebino, za pripravo katere nima lastnih virov. Proces je z dobavo podatkov uporabniku zaključen.



Slika 2.2: Proces priprave sporedov.

2.2 Prikaz sporedov

Za proces prikaza sporedov je zadolžen končni uporabnik, ki kot vhod prejme izhod procesa priprave sporedov. Ker je preostanek procesa zelo specifičen za posameznega uporabnika, se bomo omejili na predstavitev možnosti, kjer so sporedi prikazani kot jedro ali del storitve. Njihov uporabnik je gledalec, ki se bo o ogledu programa odločil na osnovi podatkov o oddaji.

2.2.1 Pregled medijev prenosa

Vhod v proces prikaza sporedov je strukturirana besedilna datoteka, ki vsebuje nek nabor podatkov. Da bi spored dospel k ciljni publiki pa potrebujemo medij preko katerega jo bo dosegel. V današnjem času prevladuje elektronski način prenosa, vendar prisotni ostajajo še vedno pomembni tiskani načini. Pri obeh lahko gledalcu prikažemo kar prejeto vhodno besedilo, a mu prav veliko ne bo koristilo, saj ni prikazano na najbolj optimalen način. Posledično si ne bo mogel ustvariti celovitega pogleda na spored, s čimer tudi prisotnost vseh možnih podatkov izgubi pomen. Prikaz sporedov v okviru elektronskega načina prenosa obravnava področje komunikacije človek-računalnik (ang. Human-Computer Interaction) [8], za prikaz v tiskani obliki pa so zadolženi predvsem grafični oblikovalci.

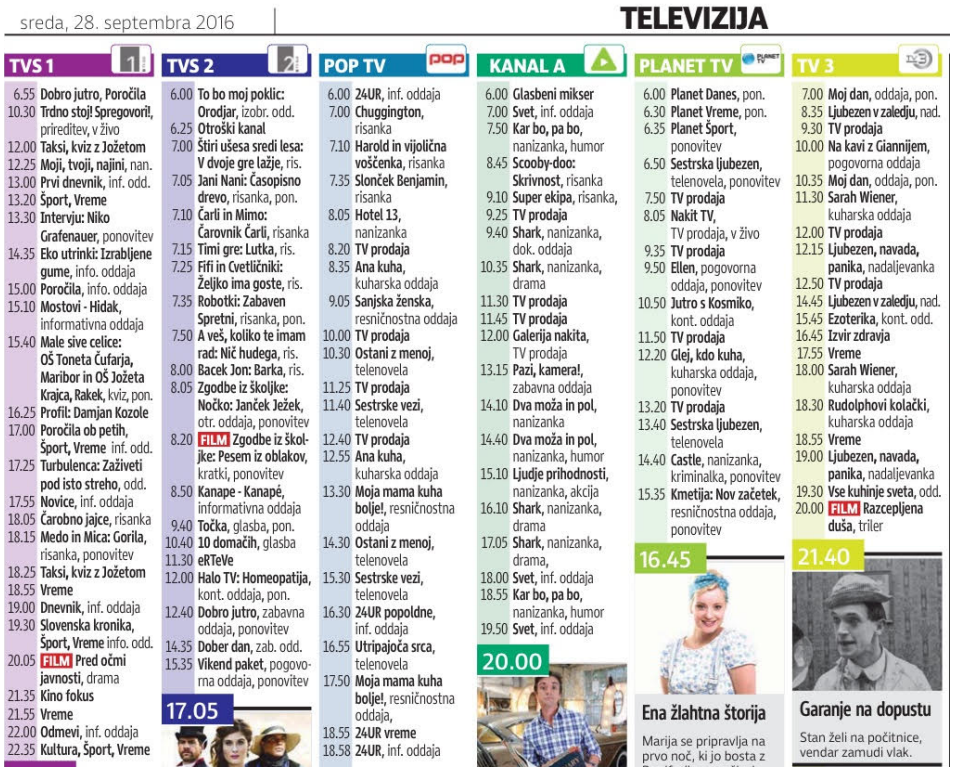
Časopisi in revije

Najzgodnejšo obliko prenosa TV sporeda k gledalcu so predstavljali tiskani načini, pod katere uvrščamo časopise in revije. Ena izmed prvih je bila ameriška *TV Guide*, ki je pričela izhajati leta 1953[1]. V časopisih je poleg glavne vsebine dodano še poglavje zanimivosti, v katero so vključeni TV sporedi za najbolj gledane kanale s strani bralcev. Časopis torej sporede vključuje kot del svoje storitve, kar velja tudi za nekatere revije. Prav tako pa še vedno obstajajo tematske revije, ki celotno vsebino prilagajajo dogajanju na TV kanalih, zaradi česar je spored jedrni del njihove storitve. Primer TV sporeda v reviji je prikazan na sliki 2.3.

Televizija

Z razvojem televizije se je v 70. letih prejšnjega stoletja uveljavil teletekst, ki je televizijam z dekoderjem omogočil prikaz tekstovnih in grafičnih informacij [11]. Tehnologija je prisotna še danes, poleg novic in ostalih vsebin pa je

[1] Michael Logan, *TV Guide Magazine's 60th Anniversary: How Desi Arnaz Jr. Became Our First Cover Star*. URL: `http://www.tvguide.com/news/tv-guide-magazine-60-arnaz-1063463/`. [Dostopano: 6.10.2017]



sreda, 28. septembra 2016 | TELEVIZIJA

TVS 1
6.55 Dobro jutro, Poročila
10.30 Trdno stoj! Spregovori!, prireditev, v živo
12.00 Taksi, kviz z Jožetom
12.25 Moji, tvoji, najini, nan.
13.00 Prvi dnevnik, inf. odd.
13.20 Šport, Vreme
13.30 Intervju: Niko Grafenauer, ponovitev
14.35 Eko utrinki: Izrabljene gume, info. oddaja
15.00 Poročila, info. oddaja
15.10 Mostovi - Hidak, informativna oddaja
15.40 Male sive celice: OŠ Toneta Ćufarja, Maribor in OŠ Jožeta Krajca, Rakek, kviz, pon.
16.25 Profil: Damjan Kozole
17.00 Poročila ob petih, Šport, Vreme inf. odd.
17.25 Turbulenca: Zaživeti pod isto streho, odd.
17.55 Novice, inf. oddaja
18.05 Čarobno jajce, risanka
18.15 Medo in Mica: Gorila, risanka, ponovitev
18.25 Taksi, kviz z Jožetom
18.55 Vreme
19.00 Dnevnik, inf. oddaja
19.30 Slovenska kronika, Šport, Vreme info. odd.
20.05 FILM Pred očmi javnosti, drama
21.35 Kino fokus
21.55 Vreme
22.00 Odmevi, inf. oddaja
22.35 Kultura, Šport, Vreme

TVS 2
6.00 To bo moj poklic: Orodjar, izobr. odd.
6.25 Otroški kanal
7.00 Štiri ušesa sredi lesa: V dvoje gre lažje, ris.
7.05 Jani Nani: Časopisno drevo, risanka, pon.
7.10 Čarli in Mimo: Čarovnik Čarli, risanka
7.15 Timi gre: Lutka, ris.
7.25 Fifi in Cvetličniki: Željko ima goste, ris.
7.35 Robotki: Zabaven Spretni, risanka, pon.
7.50 A veš, koliko te imam rad: Nič hudega, ris.
8.00 Bacek Jon: Barka, ris.
8.05 Zgodbe iz školjke: Nočko: Janček Ježek, otr. oddaja, ponovitev
8.20 FILM Zgodbe iz školjke: Pesem iz oblakov, kratki, ponovitev
8.50 Kanape - Kanapé, informativna oddaja
9.40 Točka, glasba, pon.
10.40 10 domačih, glasba
11.30 eRTeVe
12.00 Halo TV: Homeopatija, kont. oddaja, pon.
12.40 Dobro jutro, zabavna oddaja, ponovitev
14.35 Dober dan, zab. odd.
15.35 Vikend paket, pogovorna oddaja, ponovitev
17.05

POP TV
6.00 24UR, inf. oddaja
7.00 Chuggington, risanka
7.10 Harold in vijolična voščenka, risanka
7.35 Slonček Benjamin, risanka
8.05 Hotel 13, nanizanka
8.20 TV prodaja
8.35 Ana kuha, kuharska oddaja
9.05 Sanjska ženska, resničnostna oddaja
10.00 TV prodaja
10.30 Ostani z menoj, telenovela
11.25 TV prodaja
11.40 Sestrske vezi, telenovela
12.40 TV prodaja
12.55 Ana kuha, kuharska oddaja
13.30 Moja mama kuha bolje!, resničnostna oddaja
14.30 Ostani z menoj, telenovela
15.30 Sestrske vezi, telenovela
16.30 24UR popoldne, inf. oddaja
16.55 Utripajoča srca, telenovela
17.50 Moja mama kuha bolje!, resničnostna oddaja,
18.55 24UR vreme
18.58 24UR, inf. oddaja

KANAL A
6.00 Glasbeni mikser
7.00 Svet, inf. oddaja
7.50 Kar bo, pa bo, nanizanka, humor
8.45 Scooby-doo: Skrivnost, risanka
9.10 Super ekipa, risanka,
9.25 TV prodaja
9.40 Shark, nanizanka, dok. oddaja
10.35 Shark, nanizanka, drama
11.30 TV prodaja
11.45 TV prodaja
12.00 Galerija nakita, TV prodaja
13.15 Pazi, kamera!, zabavna oddaja
14.10 Dva moža in pol, nanizanka
14.40 Dva moža in pol, nanizanka, humor
15.10 Ljudje prihodnosti, nanizanka, akcija
16.10 Shark, nanizanka, drama
17.05 Shark, nanizanka, drama,
18.00 Svet, inf. oddaja
18.55 Kar bo, pa bo, nanizanka, humor
19.50 Svet, inf. oddaja
20.00

PLANET TV
6.00 Planet Danes, pon.
6.30 Planet Vreme, pon.
6.35 Planet Šport, ponovitev
6.50 Sestrska ljubezen, telenovela, ponovitev
7.50 TV prodaja
8.05 Nakit TV, TV prodaja, v živo
9.35 TV prodaja
9.50 Ellen, pogovorna oddaja, ponovitev
10.50 Jutro s Kosmiko, kont. oddaja
11.50 TV prodaja
12.20 Glej, kdo kuha, kuharska oddaja, ponovitev
13.20 TV prodaja
13.40 Sestrska ljubezen, telenovela
14.40 Castle, nanizanka, kriminalka, ponovitev
15.35 Kmetija: Nov začetek, resničnostna oddaja, ponovitev
16.45
Ena žlahtna štorija
Marija se pripravlja na prvo noč, ki jo bosta z

TV 3
7.00 Moj dan, oddaja, pon.
8.35 Ljubezen v zaledju, nad.
9.30 TV prodaja
10.00 Na kavi z Giannijem, pogovorna oddaja
10.35 Moj dan, oddaja, pon.
11.30 Sarah Wiener, kuharska oddaja
12.00 TV prodaja
12.15 Ljubezen, navada, panika, nadaljevanka
12.50 TV prodaja
14.45 Ljubezen v zaledju, nad.
15.45 Ezoterika, kont. odd.
16.45 Izvir zdravja
17.55 Vreme
18.00 Sarah Wiener, kuharska oddaja
18.30 Rudolphovi kolački, kuharska oddaja
18.55 Vreme
19.00 Ljubezen, navada, panika, nadaljevanka
19.30 Vse kuhinje sveta, odd.
20.00 FILM Razcepljena duša, triler
21.40
Garanje na dopustu
Stan želi na počitnice, vendar zamudi vlak.

Slika 2.3: Del sporeda iz slovenske TV revije STOP.

zaporedje nekaj strani namenjenih tudi prikazu sporeda. Vsebino teleteksta zagotavlja in ureja TV hiša, ki upravlja kanal na katerem je ta omogočen. Teletekst predstavlja najzgodnejšo obliko elektronskega načina prenosa TV sporedov k gledalcu. Primer strani teleteksta je prikazan na sliki 2.4. Novodobno zamenjavo za teletekst predstavlja hibridna televizija (ang. Hybrid Broadcast Broadband TV), ki v sklop vsebine prav tako vključuje spored, a je dostop do nje zaenkrat omejen zgolj na prizemno (ang. terrestrial), kabelsko in satelitsko TV omrežje.

Na televizijo so prav tako pomembno vplivali začetki interneta, ki so povzročili razvoj interaktivne televizije [10]. Za njen pričetek lahko zasluge pripišemo tudi teletekstu, ki je vnesel ločitev prikaza strani od video vsebine. To je osnova za menijsko strukturo ITV (ang. Interactive Television), kot jo poznamo danes. Funkcionalnosti, ki jih štejemo kot del interaktivne televizije

Slika 2.4: Stran teleteksta nacionalne televizije s sporedom.

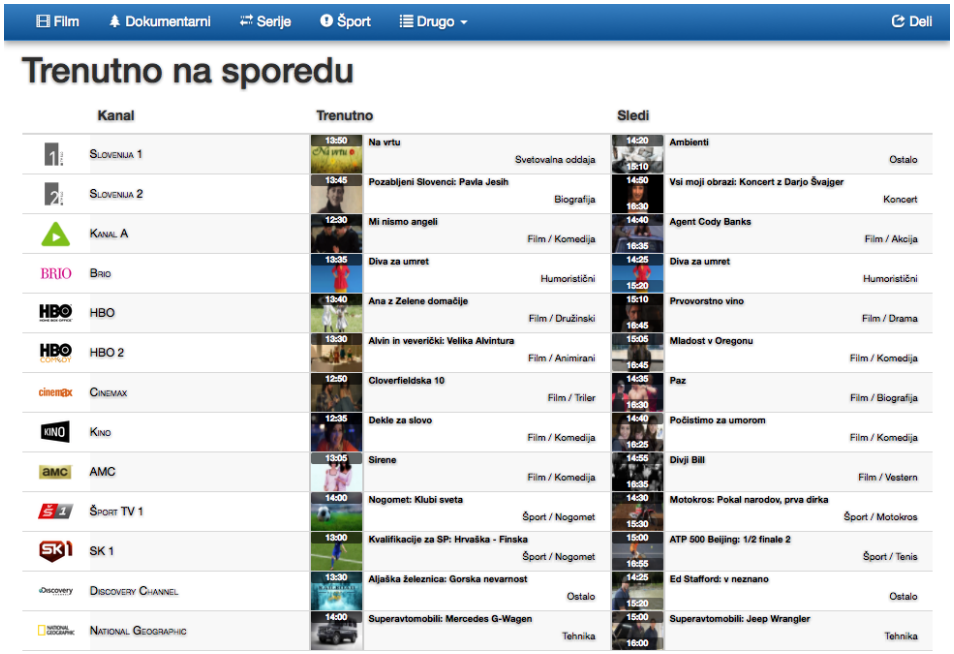
so ogled video vsebine na zahtevo (ang. Video On Demand), ki zajema tako ogled vsebine, ki ni povezana s TV kanali, kot tudi ogled pretekle vsebine (ang. time shifting), prikaz priporočil za ogled oddaj na podlagi preteklih ogledov, iskanje po oddajah, nastavljanje opomnikov in snemanje. Za podporo večjemu delu funkcionalnosti, ki delujejo na osnovi oddaj, mora biti v sistemu seveda prisoten TV spored. Posledično lahko rečemo, da je spored jedrni del interaktivne televizije.

Alternativa komercialnim načinom pretoka TV vsebine je digitalna prizemna televizija. Ta prav tako omogoča pretok vsebine in prikaz TV sporeda, le da za prikaz obojega poskrbi televizor. To pomeni, da televizor v povezavi z digitalno TV že podpira nekatere zmožnosti ITV. Interaktivna televizija je del ponudbe ponudnikov televizijskih storitev, ki za njeno delovanje naročnikom zagotovijo namenske naprave oziroma STB (ang. Set-Top Box), ločene od televizorjev. Funkcionalnosti ITV počasi prehajajo iz STB

v pametne televizorje, katerih namen je ravno neposredna integracija ITV. Za razliko od digitalne prizemne televizije, kjer za spored vsakega kanala poskrbi TV hiša, se ponudniki televizijskih storitev opirajo na ponudnike TV sporedov.

Ostale elektronske aplikacije

Dopolnitev elektronskega načina prenosa TV sporedov predstavljajo spletne in mobilne aplikacije. Te so večinoma namenjene neposrednemu spremljanju televizijskega pretoka in omogočajo okrnjeno funkcionalnost ITV. Obstajajo tudi aplikacije, ki se ne osredotočajo na zagotovitev pretoka, temveč ponujajo zgolj prikaz TV sporedov. Primer takšne aplikacije je prikazan na sliki 2.5.



Slika 2.5: Spletna aplikacija Sporedi.TV[2] omogoča podroben pregled sporedov.

[2]Sporedi.TV. URL: `http://sporedi.tv/`. [Dostopano: 25.11.2017]

2.2.2 Programski vodič

Elektronski ali interaktivni programski vodič (ang. program guide) je del interaktivne televizije, ki gledalcu na najboljši možen način prikaže TV spored. V vmesniku je navadno prikazan na povsem ločenem pogledu, ki oddaje za več kanalov razporedi v mrežno obliko glede na trenutni čas. Primer mrežnega pogleda je prikazan na sliki 2.6. Podatki o oddaji se v vmesniku velikokrat prikažejo ob preklopu na kanal, možno je tudi hitro pregledovanje oddaj za posamezni kanal.

Slika 2.6: Mrežna oblika prikaza sporeda v vmesniku večpredstavnostnega programa Kodi.

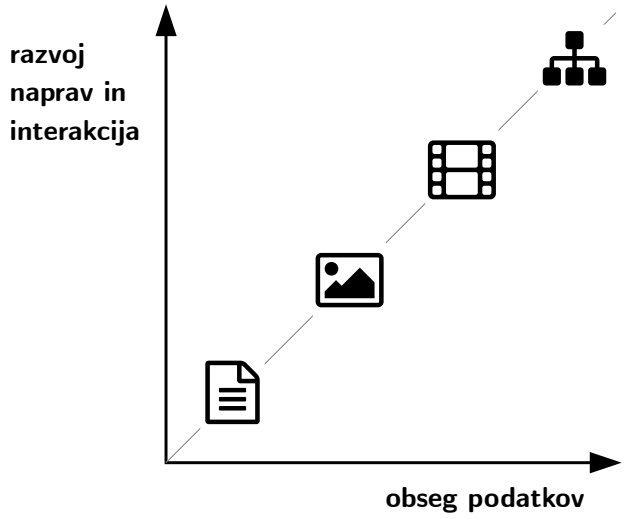
EPG (ang. Electronic Program Guide) in IPG (ang. Interactive Program Guide) pogosto obravnavamo z istim pomenom, vendar se dejansko razlikujeta [5]. EPG je izvorni izraz, ki je bil uporabljen za opis TV sporedov z manjšim naborom podatkov, kot so čas predvajanja, naslov in kratek opis. Ponujal je zgolj pregled sporeda, nad katerim uporabnik ni mogel izvršiti nobene akcije. S prihodom interaktivne televizije se je tudi pomen programskih vodičev spremenil v interaktivne. IPG je programski vodič, ki poleg podat-

kov značilnih za EPG prikazuje še obogatene podatke, hkrati pa omogoča večdnevni pregled, iskanje in priporočila oddaj iz sporeda. Namenjen je prilagajanju gledalcu v takšni obliki, da mu ni potrebno ročno iskati vsebine, ampak mu jo pripravi že naprava sama. Del funkcionalnosti ITV torej predstavlja IPG.

2.2.3 Obseg prikaza podatkov

Način prenosa določa kolikšen obseg podatkov se za prikaz potrebuje, kar je odvisno od tega kako se podatki prikazani. V tiskanih medijih nekaterih podatkov ne moremo uporabiti ali pa jih ni smiselno, saj je potrebno gledati na omejitve, ki jih prinaša tak način prenosa – na primer število strani, ki jih bo bralec še pripravljen pregledati. Pri elektronskih medijih je ravno obratno. Podatkov mora biti čim več zato, da pridobimo gledalčevo pozornost in ga poskušamo navdušiti za ogled vsebine.

Celotna televizijska izkušnja v zadnjih letih temelji na ITV. Njeni začetki so bili odvisni predvsem od učinkovitega načina prenosa podatkov, česar rezultat je bil EPG. Z uspešnim prenosom je EPG sčasoma napredoval, saj se je pojavila potreba po izboljšanju izkušnje. Z razvojem naprav in izrisa grafike se povečuje tudi potreba po interakciji [8], za katero potrebujemo podatke. Potrebe po obsegu podatkov bodo v okviru elektronskega načina prenosa zaradi tega vedno večje. Slika 2.7 prikazuje povezavo med razvojem naprav, interakcijo in obsegom podatkov.



Slika 2.7: Odvisnost od obsega podatkov se povečuje z razvojem naprav in večanjem interakcije.

Poglavje 3

Pregled ključnih tehnologij

Razvoj različnih vrst aplikacij lahko opravimo na ogromno načinov. Za vsako so najbolj priljubljena domorodna orodja, namenjena predvsem izdelavi aplikacij za izbrano platformo. Velikokrat se pojavi potreba po razvoju ene aplikacije za različne platforme, katerih glavne razvojne tehnologije se razlikujejo. Naletimo na težavo, ko za recimo dve končni platformi še lahko sami zagotovimo rešitve, za preostale zahtevane pa nimamo več znanja ali virov. Z množčnim prehodom namiznih aplikacij na splet in splošnim zanimanjem za izdelavo spletnih aplikacij se povečujejo tudi zahteve po že integriranih funkcionalnostih v obstoječe tehnologije. Vse več je uporabe spletnih tehnologij v sklopu ostalih vrst aplikacij, kot so mobilne in namizne. To pomeni, da je ključno le poznavanje spletnih tehnologij, pri čemer ni potrebno vedeti podrobnosti o končni platformi, za katero razvijamo. S tem lahko poleg aplikacij za dve končni platformi, podpremo tudi preostale, če zanje seveda obstaja integracija. Velik del platform je že podprt, prav tako so na voljo številne aplikacije, ki so narejene po opisanem principu. Kljub vsej podpori pa so še vedno odprta vprašanja v zvezi z učinkovitostjo delovanja napram domorodnim tehnologijam.

V poglavju se bomo osredotočili na pregled tehnologij in njihovih funkcionalnosti, ki so v zadnjem času med najbolj uporabljenimi. Nekatere izmed njih so nastale kot odziv na težave tehnologij uporabljenih v preteklosti in

uvajajo povsem nove koncepte na področje spletnih tehnologij. Najprej bomo na kratko obravnavali zgodovino, ki je vplivala na današnje stanje tehnologij, nato pa opisali najpomembnejše uporabljene tehnologije.

3.1 Spletne tehnologije in aplikacije

Začetki spletnih tehnologij segajo v dobo izuma svetovnega spleta, katerega glavni namen je bila izmenjava informacij s pomočjo povezav [6]. Glavna tehnologija, ki je temelj spleta še danes, je bil označevalni jezik HTML. Z njim se je pričelo obdobje izdelave **statičnih spletnih strani**, ki so bile namenjene zgolj predstavitvi informacij. Zaradi potrebe po ločitvi oblikovnega dela strani je kmalu zatem nastal CSS (ang. Cascading Style Sheets) [7], pozneje pa zaradi interakcije z uporabnikom še JavaScript [14].

Splet se je razvijal v smeri komunikacije med ponudnikom informacij in uporabnikom. Nastali so različni programski in skriptni jeziki, ki so omogočali izdelavo **dinamičnih spletnih strani**. Glavni namen teh je bil, da se uporabniki odzovejo na podane informacije ali pa jih uredijo. Bistvena razlika med statičnimi in dinamičnimi spletnimi stranmi je v načinu priprave. Statične spletne strani so že pripravljene za takojšnji izris, medtem ko se dinamične pred tem na osnovi nekih podatkov še zgenerirajo. Problem, ki nastane pri slednjem postopku je obremenitev strežnika, ki to počne. Prav tako se z vsako zahtevo po prikazu strani, ta v celoti ponovno zgenerira.

Delno rešitev za problem ponovnega generiranja so predstavljale **strani s spreminjajočimi se deli**. Te so lahko statične ali dinamične. V primerjavi s preteklima vrstama spletnih strani, dotične namesto vsakokratnega ponovnega izrisa uporabljajo tehnologije, kot je AJAX (ang. Asynchronous JavaScript and XML). Omogočile so nadomestitev obstoječih predelov strani s tistimi, ki so bile pridobljene kot odgovor na zahtevo spletnemu strežniku. Za spreminjanje strani je zadolžen JavaScript. Rezultat tovrstnih strani se je izkazal v zmanjšani obremenitvi strežnikov in boljši uporabniški izkušnji, saj uporabnikom ni potrebno čakati na ponoven izris celotne spletne strani.

V zadnjem desetletju so se spletne tehnologije razvile do te mere, da ogromno strani uporablja predhodno opisan način, saj je dobro podprt v vseh odjemalcih. Kljub temu se potreba po še boljši uporabniški izkušnji veča in dolgo nalaganje posameznih podstrani povzroča ravno nasproten učinek. Kot odgovor na ta problem so se pojavile **enostranske aplikacije** (ang. Single-Page Application – SPA), ki so spremenile miselnost predhodnih vrst strani. Razlika med njimi je v tem, da predhodne hranijo celotno poslovno logiko na strežniku, medtem ko enostranske aplikacije del ali večino logike prenašajo v sklop odjemalca. Aplikacija kot taka za svoje delovanje ni več povsem odvisna od spletnega strežnika, vendar je ta namenjen le zagotovitvi datotek za izvajanje aplikacije in morebiti podatkom, ki jih uporabnik potrebuje. JavaScript odslej predstavlja jedro enostranskih aplikacij, saj skrbi za izvajanje aplikacijske logike.

Enostranske aplikacije predstavljajo veliko težavo spletnim pajkom (ang. web crawler), ki običajno ne podpirajo JavaScripta in so odvisni od strukturnih elementov spletne strani. Zaradi prednosti, ki jih prinašajo SPA in nadomeščanjem odsotnosti začetne strukturiranosti strani, se začenjajo uveljavljati **kombinirane spletne aplikacije**. Namen teh je omogočiti, da se ob začetnem nalaganju spletne strani naloži celotna struktura, ki je prikazana uporabniku. Z nadaljnjimi akcijami se aplikacija obnaša kot enostranska. Tako spletni pajki prepoznajo vsebino strani, uporabniki pa še naprej uživajo v delu znotraj odzivne aplikacije.

Podvejo SPA predstavljajo **hibridne aplikacije**. Njihov namen je deloma predstavljen že v uvodnem delu poglavja – s pomočjo spletnih tehnologij prodreti na preostale platforme. Razvoj poteka enako, kot pri SPA, le da je razvijalec postavljen v okolje, kjer mora čimbolje izkoristiti funkcionalnosti končne platforme. Pri tem upošteva obnašanje domorodnih aplikacij. V okviru hibridnih aplikacij se omenjajo samo mobilne aplikacije, vendar zaradi načina delovanja vanje umeščamo tudi namizne aplikacije, ki delujejo na podoben način. Za izvajanje tovrstnih aplikacij je na mobilnih napravah zadolžen gradnik za prikazovanje spletne vsebine (ang. web view), na

namiznih pa je kot del aplikacije vključen spletni brskalnik. Hibridne aplikacije za izrabo domorodnih funkcionalnosti uporabljajo programske vmesnike platforme, ki so razvijalcem na voljo na višjem nivoju, kot del razvojnega paketa.

Alternativo hibridnim aplikacijam predstavljajo **napredne spletne aplikacije** (ang. Progressive Web Apps – PWA). Trenutno se razlikujejo predvsem po zmožnostih uporabe funkcionalnosti naprave[1], velik problem pa zaenkrat predstavlja še razširjenost na več platformah[2,3]. Tako kot hibridne aplikacije, tudi napredne aplikacije delujejo na osnovi gradnika za prikazovanje spletne vsebine. Ta s pomočjo zmožnosti HTML 5 poskrbi za uporabo domorodnih funkcionalnosti naprave.

3.2 HTML, CSS in SASS

HTML je označevalni jezik, ki ga uporabljamo za zapis strukture spletnih strani. Sestavljajo ga različni strukturni elementi, ki jih imenujemo značke in predstavljajo določen pomen v celotnem dokumentu. Ta je osnovan na drevesni strukturi, kar pomeni da značke skupnih predelov strani gnezdimo. Izris HTMLja opravijo brskalniki, ki jim ta predstavlja kjučen vir na podlagi katerega nato naložijo še nadaljnja sredstva, s katerimi lahko stran prikažejo v celoti. Po izrisu dokument HTML postane objektni dokument, ki je del objektnega modela dokumenta (ang. Document Object Model – DOM), nad katerim lahko izvajamo različne operacije. S HTML 5 smo dobili dodatne funkcionalnosti, ki omogočajo integracijo spleta z domačim okoljem platforme, na kateri spletna aplikacija deluje.

[1] What Web Can Do Today. URL: `https://whatwebcando.today/`. [Dostopano: 8.10.2017]

[2] Jason Grigsby, *Apple Starts Work on Progressive Web Apps*. URL: `https://cloudfour.com/thinks/apple-starts-work-on-progressive-web-apps/`, 2017. [Dostopano: 8.10.2017]

[3] Paul Thurrott, *This is What Microsoft Said About Progressive Web Apps at Build*. URL: `https://www.thurrott.com/windows/windows-10/116101/microsoft-said-progressive-web-apps-build`, 2017. [Dostopano: 8.10.2017]

CSS predstavlja sintaksna pravila za oblikovanje strukturnih elementov. Določa način izbiranja elementov in standardne opise, ki na strani odjemalca omogočajo njihovo oblikovanje. CSS med drugim uporabljamo skupaj s HTML, saj nam omogoča, da oblikovno plat dokumenta ločimo od strukturne. CSS 3 poleg klasičnih oblikovnih pravil v sklop predlog dodaja še animacije in prehode ter podporo za prilagajanje velikosti oken.

SASS (ang. Syntactically Awesome Style Sheets) nadgrajuje pravila in opise, ki jih definira CSS s standardnimi gradniki, ki jih poznamo iz programskih jezikov. V sklop sintakse dodaja spremenljivke, operatorje, pogojne izraze, zanke in funkcije. Datoteke z definiranimi pravili ne moremo uporabiti neposredno v dokumentu, vendar jo je pred tem potrebno prevesti v CSS. Kot razširitev ne omogoča dodatnih zmožnosti izven okvira CSS, ampak je namenjena predvsem boljši strukturi, berljivosti in enostavnosti definiranja oblik elementov.

3.3 JavaScript in TypeScript

JavaScript je programski jezik, ki se je sprva uveljavil na področju razvoja spletnih strani, kjer se uporablja skupaj s HTML. Omogoča dinamično dodajanje, spreminjanje in odstranjevanje elementov dokumenta HTML. Posebnost JavaScripta je, da deluje na osnovi asinhronega pristopa – ob končanju ene naloge, lahko izvedemo naslednjo, medtem pa se nadaljuje izvajanje preostalih nalog v zaporedju. Za svoje delovanje uporablja samo eno uporabniško nit. Z razvojem zalednih spletnih tehnologij, je del teh postal tudi JavaScript, saj s svojim asinhronim modelom učinkovito rešuje težave, ki jih povzročajo dolgotrajne operacije.

TypeScript je razširitev JavaScripta, ki omogoča klasičen pristop k programiranju, kot ga uvajajo drugi objektni programski jeziki, na primer Java ali C#. K standardni sintaksi dodaja statične podatkovne tipe, kar pomeni da po deklaraciji tipa, spremenljivke ne moremo inicializirati z drugim podatkovnim tipom. Poleg tega prinaša podporo za generične funkcije, de-

dovanje, deklaracijo objektov in dekoratorje, ki omogočajo izvajanje operacij nad objekti izven neposrednega konteksta. TypeScript ni zamenjava za JavaScript, ampak dodatek, ki pripomore k boljši strukturiranosti, berljivosti in zedinjenemu pristopu pri programiranju v JavaScriptu. Pred izvajanjem je TypeScript potrebno prevesti (ang. transpile) v standardni JavaScript in po možnosti ustvariti polifil (ang. polyfill), ki je uporabljen za podporo novejšim funkcionalnostim v okoljih s starejšo tehnologijo.

3.4 Enostranske aplikacije

Enostranske aplikacije, bolje poznane pod kratico SPA, so aplikacije, ki se naložijo v končno okolje (na primer brskalnik) in se med uporabo ne osvežujejo v celoti [12]. Tipična enostranska aplikacija je sestavljena iz posameznih komponent, ki se med delovanjem osvežujejo ali spreminjajo [9]. Začetki SPA segajo v obdobje dinamičnih spletnih strani, ko so se pojavile v obliki vstavljenih (ang. embed) elementov in so za izvajanje potrebovale namestitev vtičnikov. Težave tovrstnih aplikacij so bile ravno z nameščanjem dodatkov, saj ni bilo mogoče zagotoviti, da bodo vsi uporabniki že imeli nameščene – zaradi vpliva na uporabniško izkušnjo se SPA niso tako razširile. Napredek in poenotenje standardov brskalnikov sta počasi prišla na nivo, kjer se je zdelo smiselno razširiti vpliv JavaScripta na upravljanje in prikazovanje strani v celoti neodvisno od generiranja s pomočjo strežnika. Pričelo se je obdobje enostranskih aplikacij, ki zares izboljšujejo uporabniško izkušnjo s hitrostjo nalaganja. Trdimo lahko da so SPA že uveljavljena tehnologija, ki pa se še vedno izboljšuje in razširja.

Zanimanje za izdelavo enostanskih aplikacij se povečuje, vendar je pred odločitvijo o tem, ali bo aplikacija res tako zasnovana, potrebno dobro premisliti. Vprašanje, ki si ga zastavimo pred izdelavo takšne aplikacije je, kaj pridobimo v primerjavi z ostalimi načini izdelave spletnih aplikacij. Potrebno se je zavedati, da so SPA samostojne aplikacije in lahko postanejo zelo kompleksne, če se razvoja ne lotimo pravilno. Posledično to lahko vpliva tudi

na hitrost izvajanja in velikost distribuirane različice aplikacije. Navdušenje spletnih razvijalcev nad enostranskimi aplikacijami je potrebno nekoliko prezreti in dobro razmisliti ali model aplikacije res sovpada z idejo SPA.

Prednosti enostranskih aplikacij so predvsem:

- hitrost delovanja aplikacije kot celote
- ohranjanje stanja aplikacije skozi celoten življenski proces na odjemalski strani
- logika odjemalca je povsem ločena (ang. decoupled) od logike zaledja

Nekaj slabosti SPA:

- večje aplikacije običajno zahtevajo daljši začetni čas nalaganja
- aplikacija na strani odjemalca lahko zahteva izvajanje poslovne logike, ki je ne bi smeli razkriti
- potreben dodatni napor za pravilno prikazovanje vsebine, ki ugaja spletnim pajkom

3.5 RxJS

RxJS je knjižnica, ki v JavaScript prinaša podporo reaktivnemu programiranju. Gre za koncept, ki omogoča, da se na podlagi nekega dogodka lahko odzovemo na spremembe. Dogodek predstavlja asinhrona operacija, ki po zaključku sproži vse ostale operacije, ki so od nje odvisne. Vse operacije se izvajajo v obliki tokov in implementirajo vzorec opazovalca (ang. observer pattern) ter vzorec iteratorja (ang. iterator pattern). Za obdelavo tokov uporabljamo operatorje, zelo podobne tistim iz funkcijskega programiranja. Namen knjižnice je omogočiti, da se v sklopu različnih funkcionalnosti odzovemo na isti dogodek brez dodatne režije.

RxJS sestavljajo naslednji koncepti [4]:

- **opazovanec** (ang. observable) – dogodek ovit v tok, ki pošilja neke podatke
- **opazovalec** (ang. observer) – sprejema podatke opazovanca in se nanje odziva
- **naročnina** (ang. subscription) – omogoči sprejemanje odzivov opazovalca in preklic nadaljnjega sprejemanja
- **operatorji** (ang. operators) – nudijo zmožnosti za dodatno obdelavo tokov in njihovih podatkov
- **predmet** (ang. subject) – razpošlje podatek enemu ali več opazovalcem hkrati
- **razvrščevalniki** (ang. schedulers) – določajo kdaj oziroma na kakšen način se izvede obdelava podatkov

Posebnost RxJS je v tem, da opazovance loči na hladne (ang. cold) in vroče (ang. hot). Pri hladnih nobena operacija, ki vključuje naštete koncepte ni izvedena, dokler se nanjo ne naročimo. Posledica naročnine na opazovanca je sprejemanje vseh podatkov, ki jih operacije nad tokovi spustijo do opazovalca. Pri vročih je ravno obratno – podatki se pošiljajo že brez naročnine, operacije nad njimi pa določijo kaj bomo v naročnini prejeli.

Naročnine so aktivne vse dokler jih ne prekličemo ali pa se tok od katerih so odvisne pred tem že konča. Zaradi tega je dobro vedeti, da v kolikor v aplikaciji uporabljamo metode življenskega cikla, naročnine ustrezno prekličemo ob prenehanju njihove uporabe. V nasprotnem primeru se lahko zgodi, da se na en opazovanec naročimo večkrat in po nepotrebnem uporabljamo dodatna sistemska sredstva.

Izsek programske kode 3.1 prikazuje kako uporabimo nekatere zgoraj opisane koncepte.

```
console.log('top');

const observable = Rx.Observable.create(observer => {
  // Pošljemo podatke
  observer.next(1);
  observer.next(3);
  observer.next(5);

  // Zaključimo pošiljanje
  observer.complete();
})
.observeOn(Rx.Scheduler.async); // Nastavimo asinhrono
   obdelavo

const subscription = observable.subscribe(
  i => console.log('i:', i), // Izpišemo prejeti podatek
  e => console.error('e:', e), // Prikažemo napako
  () => console.log('done') // Izpišemo ob koncu prejemanja
);

console.log('bottom');

// Izpis je sledeč:
// top
// bottom
// i: 1
// i: 3
// i: 5
// done
```

Izsek programske kode 3.1: Enostaven primer uporabe konceptov RxJS.

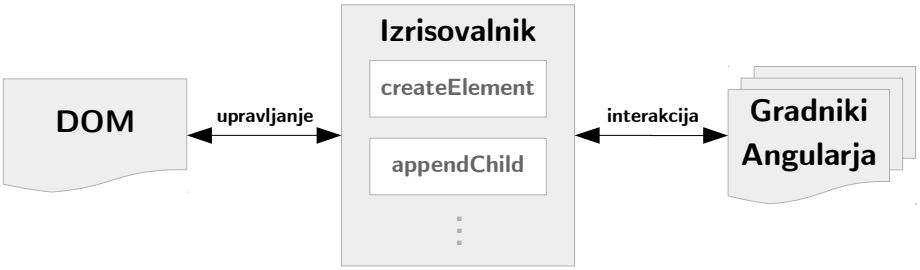
Izpis po izvedbi kode ustreza zaporedju izvajanja, saj smo opazovalcu nastavili asinhrono izvajanje.

3.6 Angular

Angular je ogrodje za izdelavo enostranskih aplikacij na osnovi spletnih tehnologij. Predstavlja skupek najboljših praks za izdelavo odzivnih spletnih aplikacij, ki jih je možno enostavno vzdrževati. Iz tega razloga je priporočena uporaba TypeScripta, v njem pa je napisano tudi jedro Angularja z vsemi pripadajočimi deli. Ogrodje se močno opira na knjižnico RxJS, saj delo s tokovi poenostavlja sledenje načelom SPA. Osredotoča se predvsem na podporo izdelavi spletnih in mobilnih aplikacij (primarno v obliki PWA), zaradi njegove večplastne arhitekture pa ga lahko prilagodimo za razvoj praktično katerekoli druge vrste aplikacije.

Bistveni koncept Angularja je modularnost. Ogrodje je povsem ločeno od končne platforme, za komunikacijo z njo pa izrablja izrisovalnik (ang. renderer). Zaradi tega je uporabniku ogrodja omogočeno, da se ne ukvarja z logiko upravljanja strukture na kateri deluje platforma, temveč poskrbi za logiko aplikacije. Poglejmo si primer delovanja na osnovi spletne aplikacije, prikazan na sliki 3.1. Platforma na kateri se stran naloži je brskalnik, ki ustvari DOM. Z JavaScriptom nato lahko do njega neposredno dostopamo in ga upravljamo. Namesto tega se pred brskalnik oziroma DOM vrine dodatna plast – izrisovalnik, ki skrbi samo za operacije v povezavi s končno platformo. Gradniki Angularja nato komunicirajo neposredno z izrisovalnikom in ne z DOM [15]. Razlog za uvedbo tega koncepta je v odcepitvi aplikacije od brskalnika. Kontekst njenega delovanja se med platformami razlikuje, zaradi česar izrisovalnik poskrbi, da je prikaz na vseh konsistenten. Primer drugačne platforme predstavlja Angular Universal[4], ki skrbi za izrisovanje prve dostopane strani aplikacije na strani strežnika. V njegovem kontekstu ne moremo uporabljati klasičnih globalnih spremenljivk brskalnika, saj v okviru strežniškega dela ne obstajajo. Omenjeno platformo lahko sicer uporabimo za pripravo kombinirane spletne aplikacije s pomočjo Angularja.

[4]Angular Universal. URL: `https://universal.angular.io/`. [Dostopano: 25.11.2017]



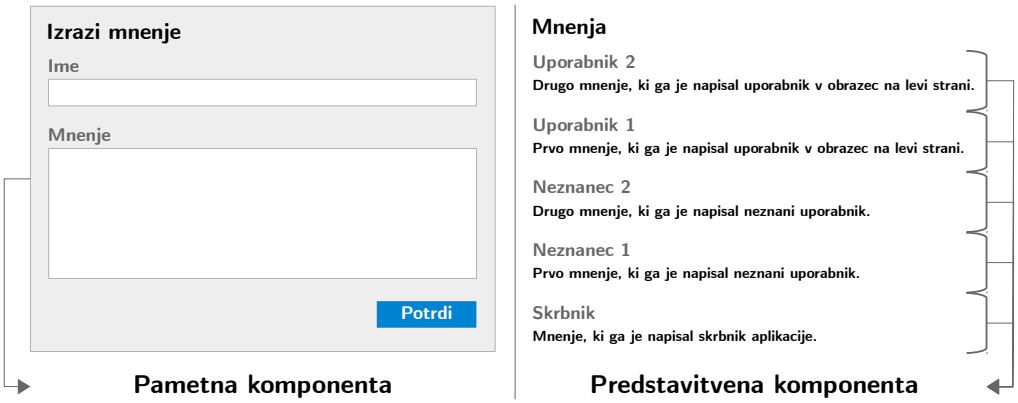
Slika 3.1: Vloga izrisovalnika med DOM in gradniki Angularja.

3.6.1 Gradniki

Angular kot ogrodje ne predpisuje stroge strukture po kateri bi aplikacije morale biti razvite, temveč ponuja gradnike s katerimi zgradimo lastno strukturo [1]. Gradnike ločimo s pomočjo meta podatkov, ki so ključni za definiranje njihovega pomena in združevanje v ločene celote. V TypeScriptu meta podatke zapišemo s pomočjo dekoratorjev.

Komponenta

Glavni gradnik, ki se izriše na končno platformo je komponenta (ang. component). Sestavljata jo vsaj logični del in predloga oziroma elementi HTML, ki skupaj tvorijo neko celoto. Razlog za povezovanje elementov v komponento je njena ponovna uporaba v drugem kontekstu. Ločimo med pametnimi (ang. smart) in predstavitvenimi (ang. presentation) komponentami. Pametne komponente vključujejo logiko, ki je pomembna iz uporabniškega vidika, kot je na primer pridobivanje podatkov iz storitve in posredovanje nadaljnji komponenti. Predstavitvene komponente so namenjene prikazu podatkov in interakciji. Kot komponente lahko opišemo skupine elementov, ki jih pogosto srečujemo na spletnih straneh – na primer navigacijski meni, iskalnik z gumbom za potrditev in obrazec za komentiranje članka. Komponente slednjega so prikazane na sliki 3.2.



Slika 3.2: Primer pametne komponente, ki združuje elemente in predstavitvene komponente, ki prikazuje podatke.

Smernica

Pomožni gradnik, ki ga uporabimo v okviru komponent je smernica (ang. directive). Omogoča sooblikovanje predlog komponent in prestrezanje dogodkov ter odzivanje nanje. Pripomore k temu, da pogosto uporabljene operacije v komponentah ločimo v svoj gradnik.

Cev

V predlogah komponent običajno izpisujemo obdelane podatke. Nemalokrat se zgodi, da je kakšnega potrebno dodatno razčleniti ali prikazati v obliki, ki ustreza uporabniku. Cev (ang. pipe) služi dodatni obdelavi podatkov, pri čemer logiko pretvorbe vhoda v izhod ločuje od komponente in omogoča njeno ponovno uporabo.

Storitev

Pomembne operacije od katerih so odvisni ostali gradniki umestimo v storitev (ang. service). Omogoča, da vso logiko, ki vpliva na interakcijo z uporabnikom umestimo v svojo plast. Ločevanje zagotovi, da spreminjanje logike vpliva na vse odvisne gradnike hkrati. Med drugim se storitve uporablja za

upravljanje s podatki, pošiljanje in odzivanje na dogodke ter izvajanje logike za pravilen prikaz vmesnika.

Modul

Vse preostale gradnike v skupino povežemo s pomočjo modula (ang. module). V njem določimo kateri gradniki delujejo kot celota ali pa že obstoječe module povežemo v glavni modul (ang. root module). Z moduli ločimo tudi funkcionalnosti vmesnika v smiselne sklope.

3.6.2 Vstavljanje odvisnosti

Angular je osnovan na tehniki vstavljanja odvisnosti (ang. dependency injection), ki omogoča, da objekte namesto nas inicializira vstavljalec odvisnosti (ang. dependency injector). Gre za ločeno ogrodje, ki je namenjeno iskanju in pripravi vseh odvisnih objektov oziroma gradnikov. Ročno inicializacijo nadomešča z namenom razširljivosti objektov in omogočanju lažjega testiranja.

3.6.3 Načini prevajanja in izvajanja aplikacij

Pri razvoju Angular aplikacije s TypeScriptom moramo pred izvajanjem kodo pretvoriti v JavaScript. Obstajata dve možnosti prevajanja. Prva je prevajanje v času izvajanja (ang. Just In Time – JIT), kjer se koda pretvori med izvajanjem aplikacije na končni platformi. Druga možnost je predčasno prevajanje (ang. Ahead Of Time – AOT), ki kodo pretvori že pred izvajanjem. Razlika med obema je v velikosti datotek, ki jih prenesemo na končno platformo in predvsem v hitrosti izvajanja. JIT običajno uporabljamo v času razvoja, saj pohitri postopek uveljavljanja sprememb, ker ne potrebuje izvesti celotnega postopka izgradnje (ang. build) aplikacije. AOT uporabimo takrat, ko želimo aplikacijo prenesti v okolje za uporabo, saj zgradi celotno aplikacijo ter zmanjša njeno velikost.

Poglavje 4

Spletna aplikacija

Z razumevanjem procesov in poznavanjem tehnologij, ki smo jih predstavili v prejšnjih poglavjih lahko nadaljujemo na enega izmed glavnih poglavij diplomske naloge. Obravnavali bomo razvoj spletne aplikacije za pripravo TV sporedov. Gre za novo aplikacijo, ki bo nadomestila že obstoječo. Razlog za prehod je dobro povzet v podpoglavju 3.1, kjer smo opisovali dinamične spletne strani, ki se generirajo na strežniku. Glavni krivec za počasnost obstoječe aplikacije je prehajanje med posameznimi stranmi, ki venomer zahtevajo generiranje celotnega vmesnika na strežniku. Cilj nove aplikacije je, da je funkcionalno primerljiva s starejšo, hkrati pa optimizira hitrost delovnega procesa, opisanega v podpoglavju 2.1. Osredotočili se bomo le na razvoj odjemalskega dela aplikacije, kjub temu pa predstavili način komunikacije z zaledjem.

4.1 Namen uporabe

Namen aplikacije je podpreti celoten proces priprave sporedov. Obstoječa aplikacija ima poleg tega še dodatne funkcije, ki nimajo neposredne povezave s procesom. Posledično morajo svoje mesto imeti tudi v novem sistemu, a jih bomo iz opisa izpustili. Preden se podamo v podrobnosti razvoja, bomo definirali sklope aplikacije in uporabniške vloge.

Aplikacija sestoji iz naslednjih sklopov:

- **spored** – omogoča pregled in urejanje sporedov, upravljanje oddaj in njihovih podatkov
- **sporočila** – služi prikazu prejetih podatkov in sredstev s strani virov podatkov
- **uporabniki** – nadzorni del aplikacije v katerem se upravlja uporabnike in njihovo pripadnost v sistemu
- **šifranti** – namenjen nastavljanju vseh infrastrukturnih podatkov, ki vplivajo na delovni proces

Obvladovanje procesa vključuje dve uporabniški vlogi:

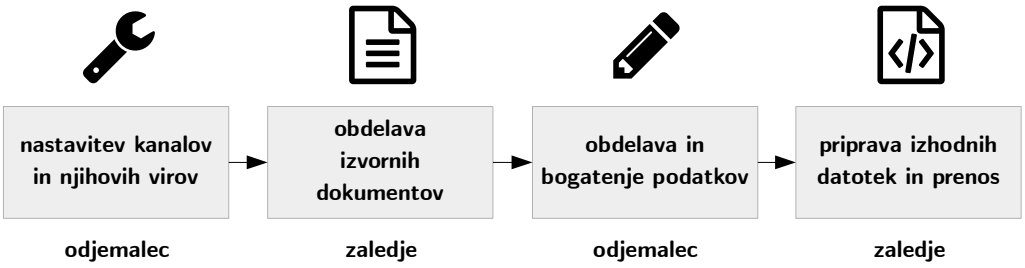
- **urednik kanalov** – predstavlja način obdelave podatkov s strani človeka, ki skrbi za pregledovanje in urejanje sporedov ter vselej spremlja morebitne spremembe s strani vira podatkov
- **skrbnik** – vzdržuje uporabniški in infrastrukturni del sistema tako, da zagotovi čimbolj nemoteno delo urednikom kanalov

Za določitev vlog uporabnikom upoštevamo naslednje omejitve. Urednik kanalov ima dostop do sklopa sporedov in sporočil. Skrbnik nadgrajuje urednikove dostope še s preostalima sklopoma uporabnikov in šifrantov.

Poleg že omenjenih ciljev spletne aplikacije moramo zagotoviti še dve stvari. Urednikom je potrebno poenostaviti uporabniški vmesnik in omogočiti, da upravljanje podatkov ter sistemska sporočila podpirajo večjezičnost.

4.2 Arhitektura in struktura aplikacije

Sistem za pripravo sporedov sestavljata dve ločeni celoti – odjemalska aplikacija in zaledje. Odjemalski del je namenjen predstavitvi in pravilnemu oblikovanju podatkov, ki jih prejema oziroma pošilja zaledju. Slednje implementira princip REST (ang. Representational State Transfer) in vključuje vso potrebno logiko za pravilno obdelovanje uporabniških podatkov. Za takšno arhitekturo smo se odločili predvsem zaradi ločevanja zahtevnih in dolgotrajnih operacij od uporabniškega vmesnika – ko sistem obdeluje podatke, uporabnik lahko nadaljuje z delom in ni odvisen od njegovega odgovora. V obstoječem sistemu je uporabnik namreč primoran počakati, da se tovrstne operacije zaključijo. Potemtakem so naloge procesa priprave sporedov razdeljene na obe celoti, kot prikazuje slika 4.1. Za vnos in urejanje podatkov poskrbi uporabnik s pomočjo odjemalske aplikacije. V našem sistemu sta zanj ključna nastavitev kanalov (v sklopu šifrantov) in obdelava podatkov (v sklopu sporeda). Na osnovi njegovega vhoda zaledje sprva pripravi podatke za obdelavo, kasneje pa jih združi v izhodno datoteko.



Slika 4.1: Arhitektura spletne aplikacije, ki ločuje proces priprave sporedov na odjemalski in zaledni del.

Spletna aplikacija temelji na Angularjevi platformi brskalnika (ang. platform browser), ki skrbi za upravljanje DOM. Poleg tega upošteva strukturne smernice, ki so priporočene v okviru dokumentacije Angularja[1]. Imeniška

[1] Angular - Style Guide. URL: `https://angular.io/guide/styleguide`. [Dostopano: 20.10.2017]

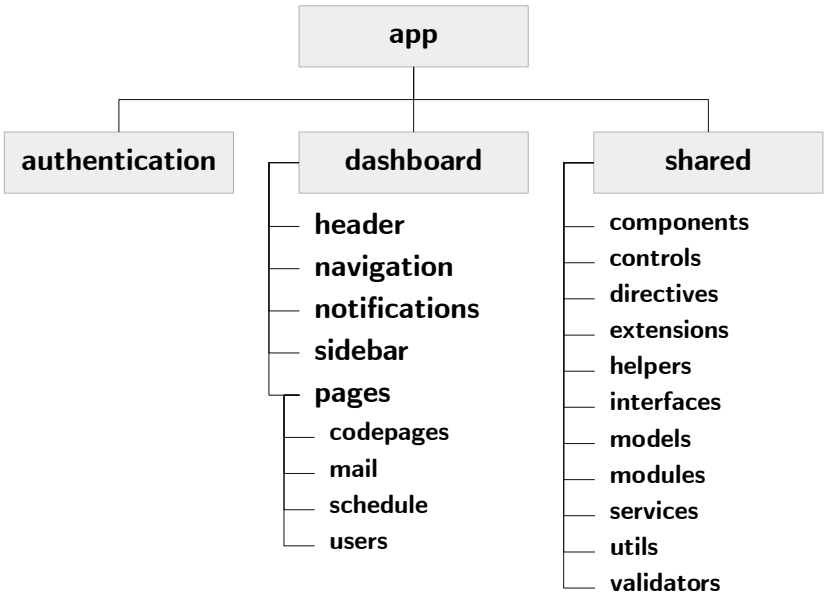
struktura (ang. directory structure) je razdeljena na enote, ki funkcionalnosti ločujejo z mapami in pomožnimi moduli. Vrhnje enote, prikazane na sliki 4.2, so:

- **overitev (ang. authentication)** – predstavlja vstopno točko v sistem, ki hkrati preverja pooblastila uporabnika za dostop do določenega sklopa aplikacije
- **nadzor (ang. dashboard)** – zagotavlja funkcionalnosti za intuitivno uporabniško izkušnjo in upravljanje s podatki, nadalje se deli na enote namenjene postavitvi strani (glava, navigacija, stranska vrstica), prikazu obvestil o izvedenih operacijah in upravljanju s skupinami podatkov v okviru podstrani
- **skupno (ang. shared)** – vsebuje vse gradnike namenjene ponovni uporabi v katerikoli enoti znotraj overitve ali nadzora (v primeru popolne ločenosti od konteksta aplikacije, se gradnike lahko ponovno uporabi v drugem projektu), nadalje se deli na komponente, nadzorne gradnike, smernice, razširitve, pomagala, vmesnike, modele, module, storitve, orodja in potrjevalnike pravilnosti

Aplikacijska struktura je zasnovana na generičnih in splošnonamenskih gradnikih. Implementirajo ključne operacije, katerim za nadaljnjo uporabo podamo podatkovne modele oziroma jih razširimo tako, da nadgradijo obstoječe funkcionalnosti. Rešitev ne sledi kakšni izmed priporočenih smernic v dokumentaciji, temveč smo ta pristop v aplikacijo vpeljali sami. Velik del aplikacije se opira na naslednje skupne gradnike:

Komunikacijska storitev (ang. API Service)

Ovojnica namenjena komunikaciji z zaledjem, uporabljena kot splošnonamenski gradnik. Predstavlja ločeno plast aplikacije s ciljem pravilnega oblikovanja prejetih in poslanih podatkov, zaznavanjem komunikacijskih napak



Slika 4.2: Imeniška struktura spletne aplikacije, ločena na več enot, ki zagotavljajo posamezne funkcionalnosti.

in vzdrževanjem overitve. Omogoča enostavno zamenjavo protokola prenosa podatkov, kot tudi konfiguracijo poti in parametrov, ki jih zaledje potrebuje za pravilno obdelovanje podatkov.

Temeljna komponenta (ang. Base Component)

Pametna komponenta, ki skrbi za komunikacijo s komunikacijsko storitvijo, uporabljena kot splošnonamenski gradnik. Zagotavlja metode za pridobivanje podatkov in omogoča izvajanje njihovih povratnih klicev glede na uspešnost prejema. Hrani stanje izvršene operacije, ki je bila posredovana storitvi. Namenjena je zagotavljanju pravilnega izrisa komponent v času pridobivanja ali pošiljanja podatkov zaledju.

Komponenta obrazca (ang. Form Component)

Abstraktna pametna komponenta, ki razširja funkcionalnosti temeljne komponente na področje upravljanja podatkov. Zaradi določanja podatkovnega

modela nad katerim se izvajajo operacije, jo uporabljamo kot generični gradnik. Namenjena je shranjevanju izvornih prejetih podatkov, beleženju narejenih sprememb nad podatki in hranjenju celotnega spremenjenega modela. V navezi s komunikacijsko storitvijo skrbi za pošiljanje spremenjenih podatkov zaledju.

4.3 Izbira dodatnih knjižnic

Razvoj aplikacije smo pohitrili z vključevanjem dodatnih knjižnic in orodij, ki smo jih pred tem skrbno izbrali. Omejili smo se na oblikovna orodja, katerih del sta knjižnica Bootstrap in ikonska pisava Font Awesome. Funkcionalna orodja predstavljajo smernice ngx-bootstrap ter knjižnici Moment.js in Cropper.js. Pri izbiri smo se izogibali vsem orodjem, ki temeljijo na drugih večjih knjižnicah, kot je jQuery. Te privzeto ne sovpadajo z miselnostjo Angularja in bi zgolj povečale velikost aplikacije. Prav tako smo med razvojem želeli vključiti že obstoječe zbirke komponent, kot sta PrimeNG in Kendo UI, a smo po tehtnem premisleku to idejo opustili. Razlog smo našli v nedodelanosti in pomanjkanju komponent za potrebe aplikacije ter znova v neželenem povečevanju velikosti aplikacije z neuporabljenimi funkcijami. Zaradi tega smo bili primorani razviti lastne nadzorne gradnike in predstavitvene komponente, kar je nekoliko upočasnilo razvoj.

4.4 Nadzorni gradniki

Uporabniško izkušnjo aplikacije lahko izboljšamo s pomočjo lastnih nadzornih gradnikov (ang. controls). Uporabniku omogočajo hitrejšo in enostavnejšo izvedbo operacij, ki jih vrši pri svojem delu. Dobro načrtovane nadzorne gradnike lahko prilagodimo več namenom uporabe, pri čemer moramo paziti na intuitivnost. V ta namen smo v okviru aplikacije razvili 11 različnih gradnikov, ki sledijo omenjenim načelom.

Angular za obvladovanje obrazcev prinaša modul *FormsModule*, ki vklju-

čuje vmesnike, katere uporabimo pri izdelavi lastnih nadzornih gradnikov. Glavni izmed njih je *ControlValueAccessor*, preko katerega implementiramo in sporočamo dogodke izvedene v gradnikih. Vsa dejanja uporabnika nad gradnikom so nato dosegljiva preko objekta, ki v osnovi razširja abstraktni razred *AbstractControl*. Običajno je to kar privzeti razred *FormControl*. Vse gradnike, ki tvorijo obrazec nato povežemo v skupino *FormGroup*. Ta hrani vrednosti gradnikov in posreduje njihove dogodke.

V nadaljevanju bomo predstavili vsak posamezni nadzorni gradnik, ki smo ga razvili in opisali način njegove uporabe. Vsakega sestavljata vsaj komponenta in modul (omogoča ponovno uporabo), po možnosti pa še dodatne podkomponente in storitve. Vsi gradniki razširjajo obstoječe gradnike HTML 5 in so uporabljeni na različnih podstraneh aplikacije. Za lažjo predstavo o načinu shranjevanja ali nadaljnji uporabi podatkov posameznega gradnika, bomo omenili podatkovni tip oziroma strukturo v kateri so predstavljeni. V aplikaciji smo opisane gradnike pogosto združili v pametne komponente. Te vključujejo logiko na podlagi katere se gradniki postavijo v določeno stanje in uporabnika vodijo skozi postopek urejanja podatkov.

Stikalni gumb

Nastavitve z dvema izbirama oblikujemo v stikalni gumb. Njegov namen je prikaz naziva nastavitve in trenuno izbrano možnost. Ima tri prikazna stanja. Prvo je določeno programsko, preostali dve pa izbere uporabnik s klikom. V onemogočenem stanju (Slika 4.3a) uporabnik ne more spremeniti njegove vrednosti in je sivo obarvan. V izključenem stanju (Slika 4.3b) je izbrana privzeta možnost, ki se nahaja desno od naziva. V vključenem stanju (Slika 4.3c) je izbrana nasprotna vrednost, prikazana na levi strani naziva. Vrednost, ki jo vrača stikalni gumb je izbrana možnost tipa `boolean`.

Iskalno vnosno polje

Privzeto vnosno polje (ang. input field) omogoča vnos besedila. Za iskanje to povsem zadošča, vendar je uporabniku potrebno nakazati, da gre za

(a) Onemogočen (b) Izključen (c) Vključen

Slika 4.3: Stanja stikalnega gumba.

iskalnik. V našem primeru je iskalno vnosno polje namenjeno takojšnjemu iskanju, brez potrebe po dodatnem potrjevanju. Posledično gradnik na vseh mestih, kjer je uporabljen deluje enako. Sestoji iz dveh prikazov. Klasičen prikaz (Slika 4.4a) vsebuje označbo mesta (ang. placeholder) in ikono lupe, ki nakazuje na iskanje. Po vnosu besedila se aktivira prikaz v času aktivnosti (Slika 4.4b), kjer se statična ikona lupe zamenja v dinamično ikono s križcem. Ob kliku nanjo se polje postavi nazaj v klasični prikaz. Vrednost, ki jo vrača iskalno vnosno polje je vneseno besedilo tipa `string`.

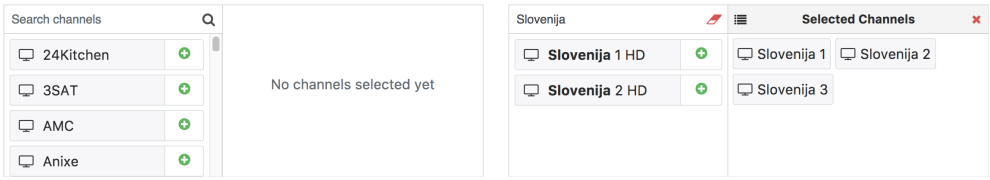
(a) Klasičen prikaz (b) Prikaz v času aktivnosti

Slika 4.4: Način prikaza iskalnega vnosnega polja.

Vlečni izbirnik

Običajna potrditvena polja (ang. checkbox) je smiselno uporabiti pri možnostih, katere mora uporabnik pregledati in se odločiti za njihovo potrditev. Trajanje pregleda pri okrog 400 možnostih zna vzeti precej časa, hkrati pa je vprašanje kako ta potrditvena polja najbolje prikazati. V ta namen smo izdelali vlečni izbirnik. Uporabniku omogoča prikaz vseh možnih opcij in iskanje preko iskalnega vnosnega polja (Slika 4.5a). Smiselno ga je uporabiti kadar uporabnik že ve katere možnosti bo izbral in mu nato prikazati izbrane v ločenem predelu (Slika 4.5b). Omogoča dva načina izbire možnosti – preko gumba za dodajanje ali s funkcijo povleci in spusti (ang. drag and drop). Vre-

dnost, ki jo vrača vlečni izbirnik je polje (tip `Array`), v katerega so vključeni identifikatorji izbranih možnosti tipa `string`.



(a) Klasičen prikaz (b) Iskalnik in prikaz izbranih

Slika 4.5: Način prikaza vlečnega izbirnika.

Nalagalnik slik

Klasični nalagalnik datotek (ang. file upload) je statični gradnik, ki hrani izbrano datoteko dokler ne sprožimo pošiljanja obrazca. To je tudi oblika v kateri je vključen v naš gradnik (Slika 4.6a). Poleg klika na gumb za iskanje datotek, podpira funkcijo povleci in spusti. V primeru ko želimo sliko urediti še pred pošiljanjem celotnega obrazca, jo je prej potrebno shraniti na strežnik. Zaradi tega smo dodali samodejno nalaganje po določenem iztečnem času (ang. timeout), prikazano na sliki 4.6b. Sproži se takoj po izbiri datoteke. Če uporabnik medtem ne prekliče nalaganja, sliko pričnemo pošiljati na strežnik. Prikaže se ikona nalaganja (Slika 4.6c). V kolikor je nalaganje uspešno, se uporabniku prikaže sporočilo o uspehu (Slika 4.6d). V nasprotnem primeru je prikazana napaka in možnost za ponovno nalaganje. Vrednost, ki jo nalagalnik slik vrača je identifikator slike tipa `string`.

Vnosno polje povezave

Povezave lahko vnesemo v običajno vnosno polje in njihovo pravilnost potrdimo na strežniški strani. Da bi prihranili večkratne tovrstne zahtevke in uporabniku pravilnost sporočili takoj po vnosu povezave, smo izdelali poseben gradnik. V primeru pravilnosti omogoča takojšen dostop do pove-

(a) Klasičen prikaz (b) Iztečni čas pred nalaganjem

(c) Prikaz v času nalaganja (d) Prikaz po uspešnem nalaganju

Slika 4.6: Način prikaza nalagalnika slik.

zave s klikom na obarvano ikono (Slika 4.7a). Če uporabnik povezave ne vnese pravilno, gradnik to zazna na podlagi regularnega izraza in se obarva rdeče (Slika 4.7b). Podpira prikaz ikon in barv za nekaj najbolj priljubljenih spletnih strani, sicer pa se prikaže generična ikona. Vrednost, ki jo vnosno polje povezave vrača je vneseno besedilo tipa `string`.

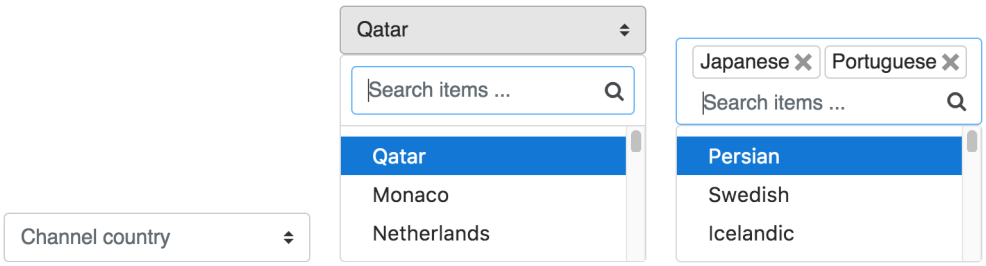
(a) Pravilna povezava (b) Nepravilna povezava

Slika 4.7: Način prikaza pravilnosti vnosnega polja povezave.

Izbirno vnosno polje

Običajno izbirno polje (ang. select field) omogoča izbiranje med več prikazanimi možnostmi. Pri več kot 10 elementih je njihov prikaz in izbiranje nekoliko oteženo. V ta namen smo pripravili izbirno vnosno polje, ki omogoča izbiro možnosti s pomočjo iskalnega vnosnega polja (Slika 4.8b). Za prepoznavanje namena polja, je to oblikovano tako kot klasično izbirno polje (Slika 4.8a). Mnogokrat se pojavi potreba po izbiri več možnosti hkrati, zaradi česar smo dodali podporo tudi temu (Slika 4.8c). Gradnik lahko nadzorujemo s tipkovnico, kar je bistvena razlika med obstoječimi gradniki in našim. Vrednost, ki jo izbirno vnosno polje vrača, je v enoizbirnem načinu

identifikator možnosti tipa `string`. V večizbirnem načinu vrača identifikatorje v isti obliki, le da v okviru polja.



(a) Klasičen prikaz (b) Ena možnost (c) Več možnosti

Slika 4.8: Način prikaza izbirnega vnosnega polja.

Večjezično vnosno polje

Vnos podatkov v več jezikih bi lahko enostavno pokrili s prikazom vnosnega polja za vsakega. Težava se pojavi kadar so možni jeziki že več kot trije, saj bi zavzeli precej uporabnega prostora. V ta namen smo združili klasično in izbirno vnosno polje (Slika 4.9a). Uporabnik v izbirnem vnosnem polju določi jezik vnosa, v klasičnem vnosnem polju pa se mu prikaže označba mesta ali vrednost, če ta že obstaja. Vrednosti drugih jezikov je mogoče prikazati s pritiskom na gumb z jezikovno ikono (Slika 4.9b). Večjezično vnosno polje vrača podatke v obliki objekta (tip `Object`), katerega ključi so jezikovne kode, vrednosti pa prevodi tipa `string`.

(a) Privzeti prikaz vnesene vrednosti (b) Razširjeni prikaz vrednosti

Slika 4.9: Način prikaza večjezičnega vnosnega polja.

Vnosno polje števil

Klasičnemu vnosnemu polju lahko nastavimo potrjevanje v obliki števil. Omogoča nam, da njegovo vrednost omejimo v določeno območje, spremenimo velikost koraka (ang. step), hkrati pa je zadovoljiva tudi podpora delovanja s tipkovnico. Kljub temu da podpira vse zahtevane funkcionalnosti, je problematičen njegov izris. Gumba za povečanje ali zmanjšanje vrednosti sta privzeto skrita in premajhna, kar je možno zaobiti s pravili CSS. Namesto tega smo raje izdelali lasten gradnik, ki ga po potrebi lahko razširimo z dodatnimi funkcionalnostmi. Slika 4.10a predstavlja njegov klasičen prikaz. Ob dosegu omejitve vrednosti se izključi gumb za povečanje ali zmanjšanje vrednosti (Slika 4.10b). Dodatna funkcionalnost, ki smo jo vgradili, je izbiranje valute preko izbirnega vnosnega polja (Slika 4.10c). Vrednost, ki jo vnosno polje števil vrača je vneseno število tipa `number`. V načinu izbire valute se vrača objekt, kjer je poleg števila vključen tudi identifikator valute tipa `string`.

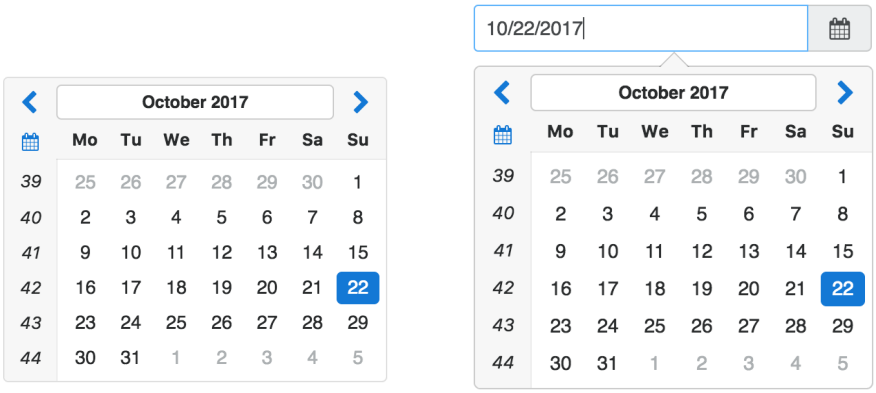
(a) Klasičen prikaz (b) Največja vrednost (c) Izbira valute

Slika 4.10: Način prikaza vnosnega polja števil.

Izbirnik datuma

Izbiranje datuma v običajnem vnosnem polju je možno. Težavo predstavlja predvsem izbirno okno, ki se izriše v domorodni obliki operacijskega sistema, hkrati pa ga ni mogoče uporabiti kot samostojni gradnik. Tako smo izdelali lasten izbirnik datuma, ki ustreza obliki aplikacije in je lahko uporabljen samostojno. V takšni obliki je prikazan na sliki 4.11a. Dodatno smo vgradili podporo za enostavnejšo izbiro po mesecih in letih. Aktivira se s klikom na mesec ali leto, ki se nahaja med puščicama za prehajanje med obdobji. V okviru načina vnosnega polja, smo dodali podporo za potrjevanje datuma, ki

ga vnese uporabnik. Poleg tega se ob kliku v polje samodejno odpre izbirnik datuma (Slika 4.11b). Za podporo funkcionalnostim, ki upravljajo z datumi, smo uporabili knjižnico Moment.js. Vrednost, ki jo vrača izbirnik datuma je neodvisna od načina prikaza, saj vedno vrne izbran datum tipa `Date`.



(a) Klasičen prikaz (b) Izbira v vnosnem polju

Slika 4.11: Način prikaza izbirnika datuma.

Izbirnik trajanja

Gradnik vnosnega polja števil bi lahko uporabili za določanje dolžine trajanja, saj omogoča vnos in korakanje po številih. Za razlikovanje vnosa trajanja od vnosa števil bi prav tako lahko podprli le še en način prikaza, a vendar je smiselno dve različni namembnosti strniti v ločena gradnika. Klasičen prikaz izbirnika trajanja je sestavljen iz vnosnega polja ter gumbov za podaljšanje oziroma skrajšanje trajanja (Slika 4.12a). Funkcionalnosti, ki jih lahko dodamo gradniku so povezane s časom. Privzeto vanj vnašamo minute (Slika 4.12b). Izbirnik trajanja vrača vrednost tipa `number`.

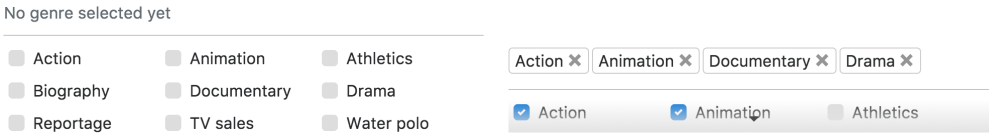
Izbirnik več možnosti

Običajna potrditvena polja uporabimo kadar lahko izberemo več različnih možnosti, pri čemer je smiselno da preverimo vse možne. V kolikor je teh

(a) Klasičen prikaz (b) Prikaz po izbiranju

Slika 4.12: Način prikaza izbirnika trajanja.

malo, ni problema z uporabo prostora. V nasprotnem primeru pa je za pregled vseh kaj kmalu potrebna uporaba drsnika (ang. scrollbox). Slika 4.13a prikazuje razpored možnosti ob prvem izbiranju med njimi. Problem, ki ga rešuje naš gradnik je namreč izpostavitev izbranih možnosti ob ponovnem izbiranju. Hkrati skrije preostale možnosti tako, da ne uporabljajo vsega prostora po nepotrebnem, temveč se prikažejo šele ob kliku na gumb za razširitev. Rešitev je prikazana na sliki 4.13b. Vrednost, ki jo izbirnik več možnosti vrača je polje, v katerem se nahajajo identifikatorji možnosti tipa `string`.



(a) Klasičen prikaz (b) Prikaz izbranih

Slika 4.13: Način prikaza izbirnika več možnosti.

4.5 Uporabnikova pot

Spletna aplikacija je primarno namenjena urednikom kanalov in skrbnikom. Posledično je postavitev strani v veliki meri prilagojena slednjima. Skrbnikov, ki bodo uporabljali spletno aplikacijo je 5, medtem ko je urednikov kanalov 30. V nadaljevanju bomo najprej opisali sestavo uporabniškega vme-

snika, nato pa se osredotočili na podstrani, ki so prvotno namenjene posamezni uporabniški vlogi.

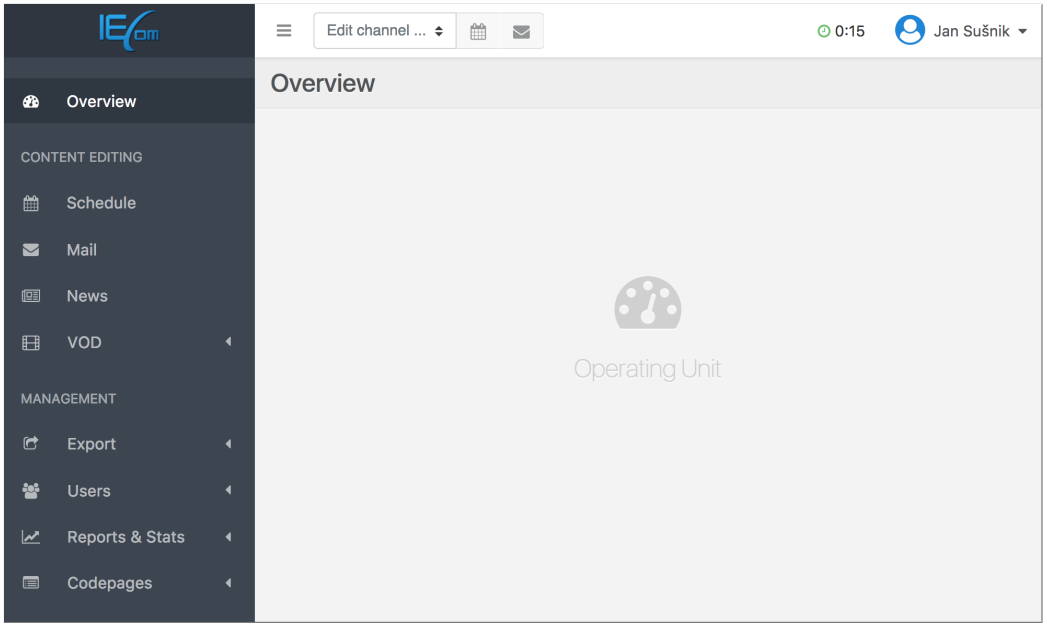
4.5.1 Uporabniški vmesnik

Prva stran, ki se prikaže na novo prispelemu uporabniku je prijavni obrazec. Spada pod enoto overitve. Dostop do vmesnika aplikacije je omejen na urednike kanalov in skrbnike. Vsak izmed njih poseduje en uporabniški račun, prepoznan po uporabniškem imenu in geslu. Po uspešni overitvi zaledje odjemalcu pošlje žeton JWT (ang. JSON Web Token)[2]. Tega odjemalec vključi v vsak nadaljnji zahtevek, ki ga pošlje zaledju. Bistvena prednost žetona JWT je v tem, da vanj lahko vključimo uporabniške podatke. Te lahko prikažemo uporabniku (na primer ime in priimek) ali pa jih uporabimo za prilagoditev vmesnika (na primer jezik). Na osnovi teh podatkov se v aplikaciji odpre glavna podstran nadzorne enote.

Nadzorna enota je razdeljena na dve podenoti – pomagalno in operacijsko. Zasnova obeh je prikazana na sliki 4.14. V pomagalno spada celotna postavitev strani, v operacijsko pa podstrani na katerih uporabniki vršijo operacije. Pomagalni del je sestavljen iz glave na vrhu strani, navigacijskega predela na levi in stranske vrstice na desni. V glavi strani so uporabniku na voljo pomagalo za pomanjšanje ali razširitev navigacijskega predela, izbirno vnosno polje za hitro izbiro kanala za urejanje, gumba za pregled sporeda kanala in prikaz njegovih sporočil v stranski vrstici, stikalo za nadzor delovne urice in uporabniški meni za dostop do nastavitev ter izhoda iz sistema. Navigacijski predel vsebuje povezave za dostop do podstrani, ki se odprejo v operacijski podenoti. Razdeljene so v sklop urejanja vsebine in sklop upravljanja sistema. Prvi je načeloma namenjen urednikom kanalov, drugi pa skrbnikom. Stranska vrstica služi hitremu prikazu podatkov, brez prehoda med različnimi operacijskimi podenotami.

V okviru pomagalne podenote so uporabljene štiri storitve: *ChannelSer-*

[2]RFC 7519 - JSON Web Token (JWT). URL: `https://tools.ietf.org/html/rfc7519`. [Dostopano: 24.10.2017]



Slika 4.14: Nadzorna enota uporabniškega vmesnika, razdeljena na pomagalno in operacijsko podenoto.

vice, *NavigationService*, *SidebarService* in *WorkClockService*. Vsaka skrbi za hranjenje podatkov in pravilen prikaz teh v vmesniku, prva in zadnja pa za svoje delovanje izrabljata tudi funkcionalnosti komunikacijske storitve. Njuni podatki se tesno opirajo na zaledje, saj so določeni s strani skrbnikov ali na podlagi preteklih dostopov do sistema.

4.5.2 Skrbnik

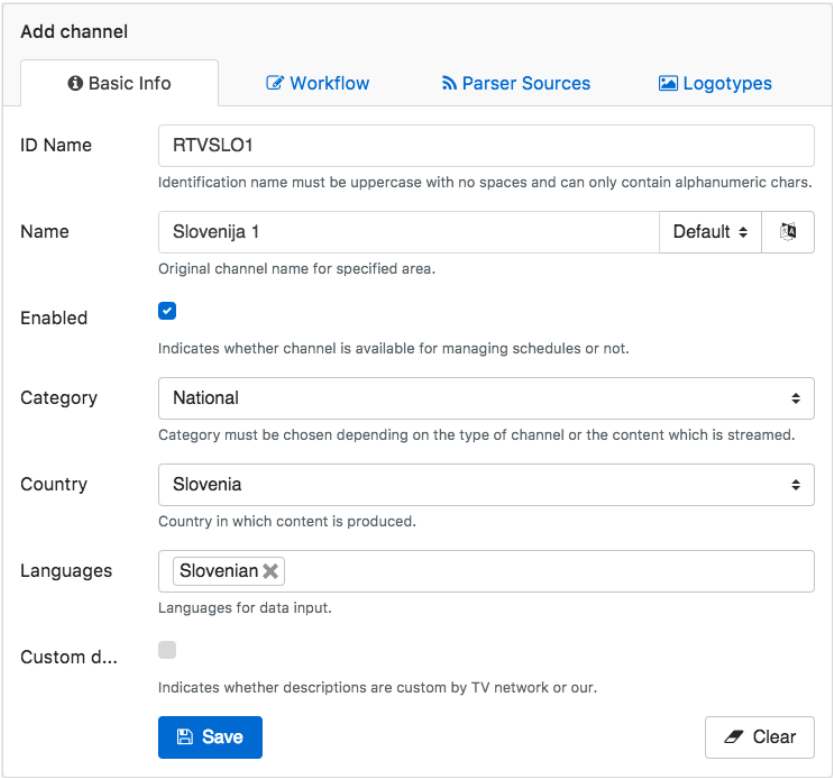
Delovne naloge skrbnika umestimo v ničti ali uvodni korak, nahajajoč se pred dejanskim pričetkom procesa priprave sporedov. Na sliki 4.1 smo ta korak opisali kot nastavitev kanalov in njihovih virov, sicer zajet v sklopu šifrantov. Poleg tega skrbnik nadzoruje še uporabniški sklop, ki je iz vidika procesa prav tako postavljen v infrastrukturni del sistema. Podstrani obeh sklopov sta sestavljeni iz podatkovne tabele z gumboma za urejanje ter brisanje, iskalnikom po vnosih in obrazca, ki omogoča urejanje podatkov.

Komponenta podstrani je osnovana na temeljni komponenti in poleg ostalih predstavitvenih komponent vključuje komponento obrazca. V nadaljevanju se bomo osredotočili le na podatkovna polja v obrazcih, saj so ostale komponente povsod iste. Opisali bomo tudi namen uporabe teh podatkov.

Sklop šifrantov zajema upravljanje vseh podatkov, ki so nujni za delovanje sklopa sporedov. Zajema v nadaljevanju opisane podstrani.

Kanali

Vključuje obrazec sestavljen iz štirih zavhikov (Slika 4.15). Prvi obravnava osnovne informacije o kanalu, kjer skrbnik izpolni ime, kategorijo in državo. Določi ali je kanal aktiven – je zanj potrebno pripravljati spored, delovne jezike v katerih morajo biti na voljo podatki sporeda in ali morajo biti prikazani opisi oddaj nujno vzeti iz izvornega dokumenta. V drugem izpolni opis procesa dela, ki pove na kakšen način se ureja spored in je namenjen uredniku kanala. Običajno opiše kaj je potrebno paziti pri kombinirani obdelavi. Nato nastavi število dni, ki določajo koliko vnaprej mora biti spored pripravljen. V tretjem zavihku vnese vir iz katerega se prenese izvorni dokument, ki ga zagotavlja vir podatkov. Na izbiro ima več načinov prenosa – preko spletne storitve, FTP, e-pošte in drugih. Zanj nato izpolni vse podatke, s katerimi je možno dostopati do vira. Po uspešni nastavitvi vira, se skrbniku prikaže komponenta za povezovanje lastnosti oziroma vozlišč dokumenta z lastnostmi v sistemu. S tem določi kateri podatek iz dokumenta mora biti prenešen v bazo pod katero lastnostjo. Tako pripravi navodila zaledju, da ta zna pravilno prebrati spored in ga umestiti v sistem. V zadnjem, četrtem zavihku, skrbnik nastavi logotip kanala, kateremu določi različico – navaden, enobarven. Podatki obrazca so uporabljeni za umestitev sporeda na posamezen kanal, definiranje delovanja zaledja in pravilno pripravo izhodnih datotek.



Slika 4.15: Zavihek z osnovnimi informacijami v okviru obrazca za upravljanje kanalov.

Kategorije kanalov

Prikazuje obrazec z enim večjezičnim vnosnim poljem, ki zahteva vnos imena kategorije. Ta je nato uporabljena na podstrani s kanali. Namen kategorij je povezati kanale s podobno vsebino v skupine.

Skupine kanalov

Ima obrazec s tremi gradniki. Skrbnik mora določiti ime in opis skupine ter izbrati kanale, ki spadajo vanjo. Razlog za delitev kanalov v skupine je možnost skupne uporabe enega vira za več kanalov.

Države in jeziki

Ločeni podstrani za države in jezike vključujeta obrazec sestavljen iz polja z imenom in polja s pripadajočo dvočrkovno kodo. Podatki se uporabijo v okviru podstrani s kanali in v sklopu sporeda.

Žanri

Predstavlja obrazec z dvema gradnikoma. Prvi je večjezično vnosno polje z imenom žanra, drugi pa potrditveno polje, ki označuje aktivnost žanra. Namenjen je uporabi na podstrani s kategorijami in v sklopu sporeda.

Kategorije

Ima obrazec s štirimi polji. Skrbnik mora podati ime ter opis in nastaviti ali je kategorija aktivna. Prisotno je polje z izbirnikom več možnosti, kjer se določi kateri žanri pripadajo kategoriji. Podatki so nato uporabljeni v sklopu sporeda.

Uporabniški sklop je namenjen določanju in omejitvi dostopov. Podstrani, ki jih vključuje so opisane v nadaljevanju.

Uporabniki

Predstavlja obrazec sestavljen iz dveh osnovnih in dveh pomožnih zavihkov. Osnovna zavihka sta namenjena vsem uporabnikom v sistemu, pomožna pa le skrbnikom in urednikom kanalov.

V prvem osnovnem zavihku mora skrbnik izpolniti uporabniško ime, geslo, e-poštni naslov, ime in priimek uporabnika ter jezik vmesnika. Določiti mora ali je uporabnik omogočen, kar vpliva na zmožnost prijave v sistem. Drugi osnovni zavihek je namenjen določanju skupin pravic in dodatnih dovoljenj uporabniku. Skrbnik lahko izbere med pripravljenimi skupinami, ki določijo sklope do katerih dostopa uporabnik. V primeru omejitve dostopa na neko podmnožico entitet sklopa, mora dodatno določiti kontekst. Izbira

lahko tudi posamezna dovoljenja za katera veljajo ista pravila, le da imajo pri upoštevanju dostopa prednost pred skupinami. Za nastavitev pravic urednikom kanalov, skrbnik izbere skupino imenovano *Urednik kanalov* ter v prikazani podkomponenti izbere kanale in pripadajoče jezike, ki jih uporabnik sme urejati. Z njimi določi kontekst, ki pove ali ima v nekem sklopu dostop do izbranih entitet. Za določitev uporabniške vloge skrbnika je postopek podoben, le da je izbira skupine *Skrbnik* splošna in ne potrebuje določanja konteksta.

V prvem pomožnem zavihku skrbnik izpolni dodatne podatke o uporabniku, kot so naslov, kraj, telefonska številka, matična številka in številka bančnega računa. Ti podatki so pomembni zaradi obračunskega sklopa aplikacije (ni zajet v opisu). Drugi pomožen zavihek je namenjen delovnim nastavitvam uporabnika. Tu skrbnik nastavi jezike v katerih uporabnik lahko ureja spored. Prikažejo se pri nastavljanju kanalov za urejanje, kjer skrbnik vidi presek uporabnikovih jezikov in jezikov kanala.

TV hiše

Prikazuje obrazec s tremi zavihki. Prvi zajema osnovne podatke o TV hiši – njen naziv in ime, telefonsko številko ter e-poštni naslov kontaktne osebe. V drugem zavihku skrbnik s pomočjo vlečnega izbirnika izbere kanale, ki pripadajo TV hiši. Podobno velja tudi za tretji zavihek, kjer mora določiti še seznam zaposlenih ter izbira med uporabniki v sistemu. Omeji se na tiste uporabnike, ki bodo imeli dostop do sistema. Razlog za obstoj te podstrani je nadzor in upravljanje sporedov s strani TV hiš.

Stranke

Ima obrazec sestavljen iz treh zavihkov. Prvi vsebuje polja z osnovnimi podatki o stranki, kjer skrbnik nastavi naziv in določi ali je omogočena. Drugi zavihek podobno kot pri podstrani TV hiš omogoča izbiro zaposlenih, ki imajo dostop do našega sistema. V tretjem zavihku skrbnik lahko upravlja generične slike. V izhodnih datotekah se pojavijo kadar za kakšno izmed od-

daj ni mogoče pridobiti prave. Podatki strank se uporabljajo v sklopu izvoza (ni zajet v opisu), ki določa način priprave izhodnih datotek in njihovega prenosa.

Dovoljenja

Predstavlja obrazec sestavljen iz dveh vnosnih polj. Prvo določa ime dovoljenja, drugo pa ključ, ki definira dostopno pot. Ime je uporabljeno le za bolj razumljiv prikaz skrbniku, medtem ko je ključ uporabljen kot atribut v kodi zaledja. Nastavljen je na metodah, kjer dotično dovoljenje uporabniku omogoča dostop. Za razliko od ostalih podstrani tu vnosa ni mogoče dodati, temveč je možno le urejanje imen dovoljenj, ki se pojavijo v kodi zaledja.

Skupine pravic

Vključuje obrazec s tremi polji. Zajema ime in opis skupine, ki sta namenjena prepoznavanju pravic s strani skrbnika. Glavno polje predstavlja nadzorni gradnik, sestavljen iz več izbirnih vnosnih polj in gumbov. V okviru tega določi dovoljenje ter vrsto dostopa – brez, branje, pisanje, brisanje, polni nadzor. V skupino umesti vsa dovoljenja, ki ustrezajo eni uporabniški vlogi.

4.5.3 Urednik kanalov

Delo urednika kanala zajema obdelavo podatkov sporeda, ki smo ga uvrstili kot drugi korak v proces priprave sporedov. Na sliki 4.1 smo ta korak opisali z obdelavo in bogatenjem podatkov, kar spada v sklop sporeda. Za doseganje ažurnosti in pridobivanje točnih podatkov, ima urednik dostop do sklopa sporočil. V njem lahko prebira e-pošto prejeto s strani virov. Vse v nadaljevanju opisane podstrani so osnovane na temeljni komponenti.

Sklop sporedov pokriva pregled in urejanje sporeda za posamezen kanal. Podstrani, ki jih vključuje so opisane v nadaljevanju.

Pregled sporedov v uredništvu

Podstran prikazuje stanje vseh kanalov, ki jih upravlja urednik. Za posamezen kanal sta izpisana datum do katerega morajo biti podatki pripravljeni in zadnji dan do katerega dejansko so pripravljeni. V kolikor je zadnji dan enak ali presega datum omejitve in so vsi podatki v ustrezni obliki, je uredniku kanal prikazan kot uspešno urejen. V primeru, da to ne drži, se mu prikaže opozorilo, ki točno pove kaj se dogaja s sporedom. Lahko se med njegovo zadnjo potrditvijo spored spremeni na podlagi strojne obdelave podatkov. Obstaja možnost, da dan ali oddaja vsebujeta napako, ki onemogoča pripravo pravilnega izhoda. Urednik kanala lahko pripravo programskega dne izpusti zaradi nedosegljivosti izvornega dokumenta, zaradi česar se mu prikaže opozorilo o manjkajočem sporedu. Podatki na tej podstrani uredniku služijo za pomoč pri sledenju napredka na vseh kanalih, tako da mu ni potrebno konstantno pregledovati posameznih sporedov.

Upravljanje sporeda za kanal

Podstran je v osnovi namenjena prikazu koledarja z oddajami za obdobje sedmih dni od izbranega datuma (Slika 4.16). Za posamezen dan so izpisane vse oddaje, pri čemer je prikazana ura predvajanja in njen naslov – klasičen prikaz je poenostavljen, saj je v dnevu povprečno 20 oddaj. Za prehajanje med programskimi dnevi je na voljo izbirnik datuma in navigacijski gumbi. Urednik prav tako lahko izbere jezik v katerem želi pregledovati spored – prikazane ima jezike, katere mu skrbnik določi za urejanje. S klikom na gumb z dodatnimi informacijami si lahko prebere opis načina dela na kanalu, ki mu ga prav tako pripravi skrbnik. Na voljo ima tudi razširjeni prikaz, ki omogoča upravljanje posameznih dni sporeda in operacije, ki se vršijo nad celotnim izbranim obdobjem.

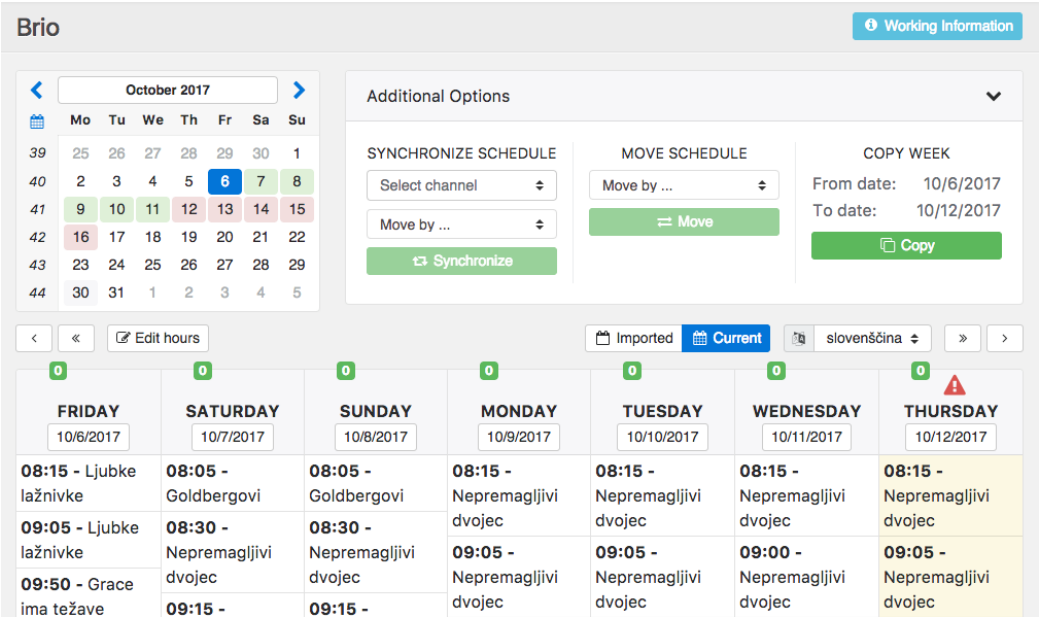
Urednik ima na izbiro dve vrsti sporeda – spored pripravljen iz izvorne datoteke in delovni spored, ki ga ureja sam. Prvi je namenjen pregledu sporeda, ki je dospel s strani vira. Drugi na podlagi povezovanja oddaj, ki ga opravi zaledje, prikazuje njihovo stanje. Oddaje, ki so bile prepoznane v

bazi se samodejno umestijo na spored, v nasprotnem primeru se obarvajo rumeno in jih mora urednik povezati ali dopolniti. Kadar se s kurzorjem postavimo nad okvir z oddajo, se prikažejo dodatne možnosti. V primeru povezane oddaje, urednik lahko pregleda njene osnovne podatke (poleg že izpisanih), jo uredi ali odstrani iz sporeda. Če je oddaja rumena, pa poleg omenjenega lahko za povezovanje izbere eno izmed predlaganih oddaj v spustnem seznamu (ang. dropdown list) ali pa klikne na gumb za iskanje. S pomočjo prikazanega okenca (ang. dialog) nato poišče oddajo in jo poveže z oddajo na sporedu.

Urednik spored ureja zaporedno po programskih dneh. Zaradi tega je nad posameznim prikazano število priporočenih oddaj (določi urednik na podstrani za upravljanje oddaje), kot tudi morebitne napake. Te razširjajo kratke opise, ki so prikazani na podstrani za pregled sporedov v uredništvu. Poleg tega sta na voljo dva dodatna gumba. Izbris vseh oddaj v dnevu in dodajanje nove oddaje. Kadar je programski dan pravilno in v celoti izpolnjen, se prikažeta še dva gumba. Kopiranje vseh oddaj dneva na drug dan – odpre se okence, v katerem urednik določi dan in po možnosti tudi drug kanal, kamor želi skopirati oddaje. Potrditev dneva – sproži pripravo izhodne datoteke, ki jo zaledje prenese k strankam ter s tem zaključi iteracijo sporeda.

Glavne operacije nad sporedom so razširjene še z dodatnimi funkcionalnostmi. V neposredni povezavi s koledarjem ima urednik možnost urejanja ur oddaj na delovnem sporedu. Vključi jo s pritiskom na gumb, ki se nahaja poleg navigacije. S tem se onemogočijo vse prikazovalne funkcije (prehajanje med programskimi dnevi, zamenjava vrste sporeda ...). Urednik se tako osredotoči na urejanje ur posameznih oddaj. V primeru zamenjave vrstnega reda oddaj v izvornem dokumentu, ima na voljo funkcijo za premikanje le-teh po celotnem koledarju s pomočjo načina vleci in spusti. Ko z urejanjem zaključi, mora spremembe potrditi ali preklicati. V okviru dodatnih funkcij so mu na voljo še sinhronizacija in premikanje sporeda ter kopiranje celotnega izbranega obdobja. Pri sinhronizaciji izbrano obdobje prenese na drug kanal,

izbere pa lahko tudi premik sporeda. Funkcija premika sporeda vse oddaje v obdobju zamakne za izbrano število ur ali dni. Kopiranje izbranega obdobja kopira vse oddaje na obdobje čez naslednjih sedem dni.



Slika 4.16: Podstran za upravljanje sporeda, ki omogoča različne operacije nad programskimi dnevi in oddajami.

Upravljanje oddaje

Podstran omogoča dodajanje ali urejanje oddaj. Do nje dostopamo preko podstrani za upravljanje sporeda za kanal, kjer izberemo dodajanje nove oddaje ali uredimo obstoječo v izbranem obdobju. Razdeljena je na tri predele, namenjene urejanju podatkov in shranjevanju. V okviru te podstrani so uporabljeni skoraj vsi nadzorni gradniki, ki smo jih opisali v podpoglavju 4.4. Posamezni obrazci znotraj podstrani temeljijo na komponenti obrazca, za njihovo enotnost pa skrbi vrhnja komponenta, v katero se umeščajo vsi podatki.

V prvem predelu urednik izpolni podatke, ki oddajo povezujejo s spo-

redom. Določi datum, čas in trajanje. Nadalje izbere tip predvajanja, s katerim označi ali gre za premiero oddaje na kanalu, poteka v živo ali pa je ponovljena. Dodatno označi ali je oddaja priporočena in v tem primeru opiše razlog priporočila. Kadar so pri predvajanju oddaje podnapisi vžgani v sliko, mora navesti tudi jezik podnapisov. Pred nadaljevanjem na naslednji predel, mora izbrati vrsto oddaje, ki je običajno film ali nadaljevanka, na voljo pa ima še nekaj drugih možnosti.

V drugem predelu se uredniku prikaže obrazec za vnos podatkov o oddaji (Slika 4.17). V osnovi prikazuje splošen nabor možnosti, ki se razširijo glede na izbrano vrsto oddaje. Za primer nadaljevanke se prikažeta še polji za vpis sezone in epizode. Obrazec je sestavljen iz devetih zavihkov, saj je podatkov ogromno in so posledično razvrščeni po kategorijah. Obveznost vnosa podatkov je odvisna od posameznega kanala in njegove nastavitve shem za stranke (ni zajeto v opisu). V splošnem zavihku najdemo podatke kot so naslov, podnaslov, jezik, kategorija, žanr, starševska ocena, leto produkcije in ostale. Zavihek ocen vsebuje ocene in povezave do nekaterih portalov, ki omogočajo ocenjevanje oddaj. V tretjem zavihku najdemo polja opisov, ki se delijo na povzetek, opis vsebine in splošni opis. Zavihek s povezavami vključuje vnosna polja s povezavami do družabnih in uradne spletne strani. V petem zavihku se nahajajo produkcijski podatki, kjer urednik uredi režiserje, igralce, komentatorje in ostale sodelujoče osebe. Zavihek urejenih slik je namenjen temu, da urednik poišče štiri slike oddaje (v primeru nadaljevanke jih šest, saj izbere še sliki sezone in epizode) v določeni resoluciji in jih izreže tako, da najbolj ustrezajo več različnim prikaznim razmerjem (ang. aspect ratio). V sedmem zavihku je prostor za ostale slike (plakati, slike igralcev in njihovih vlog), ki jih je možno pridobiti iz zunanjih virov. Zavihek video vsebin vključuje napovednike, katerih ustreznost določi urednik in po potrebi poišče dodatne oziroma jih odstrani. Zadnji, deveti zavihek zajema razširjene podatke namenjene obogatitvi vsebine. Vključuje nagrade, glasbo, snemalne lokacije, datume izida po državah in ostale.

Pri vnosu podatkov so uredniku na voljo pomagala, ki pohitrijo njegovo

Emission data

General

ORIGINAL TITLE: Lord of the rings
ORIGINAL SUBTITLE: The fellowship of the ring
SUBTITLE INDETER.

slovenščina magyar hrvatski

TITLE: Gospodar prstanov
SUBTITLE: Bratovščina prstana
SHORT TITLE: Gospodar prstanov: Bratovščina prstana
Remaining characters: 2

PRIMARY LANGUAGE: English
CATEGORY: Film
PRIMARY GENRE: Fantasy
OTHER GENRES: Adventure, Drama

OTHER LANGUAGES: Other Languages
COUNTRIES OF PRODUCTION: USA

PARENTAL RATING: Over 15 years (15)
RUNTIME (MINUTES): 175
PRODUCTION YEAR: 2001

BUDGET: 93000000 $ (US)
OPENING WEEKEND: 844711 $ (US)
GROSS: 315544750 $ (US)

Ratings

General Ratings Descriptions Links Production Data Edited images Other images Video content Premium content

Slika 4.17: Obrazec za urejanje podatkov o oddaji, razdeljen na devet zavihkov. Vsebuje polja, ki jih je potrebno izpolniti za film.

delo. Pri naslovu in podnaslovu oddaje je vmesnik razdeljen na zavihke z jeziki, kjer je dosegljiv vsak pripadajoči naslov. Podobno velja tudi za opise, kjer poleg tega urednik lahko izbere tudi jezik opisa, ki se mu prikaže ob vnosnem polju in omogoči prevajanje le-tega. Med vnosom izvornega naslova se prikazujejo predlogi iz baze, s klikom na lupo poleg polja, se izvede še iskanje po zunanjih virih. Rezultati so nato prikazani v okencu, kjer urednik izbere ujemajočo se oddajo. Za urejanje slik se prikaže okence v katerem urednik najprej preko nalagalnika slik naloži slikovno datoteko. Nato jo s pomočjo orodja za izrezovanje, ki temelji na knjižnici Cropper.js, tudi dokončno obdela. Zadnje pomagalo je namenjeno iskanju video napovednikov, ki se prikaže v okencu. V njem urednik poišče relevantne posnetke, katere

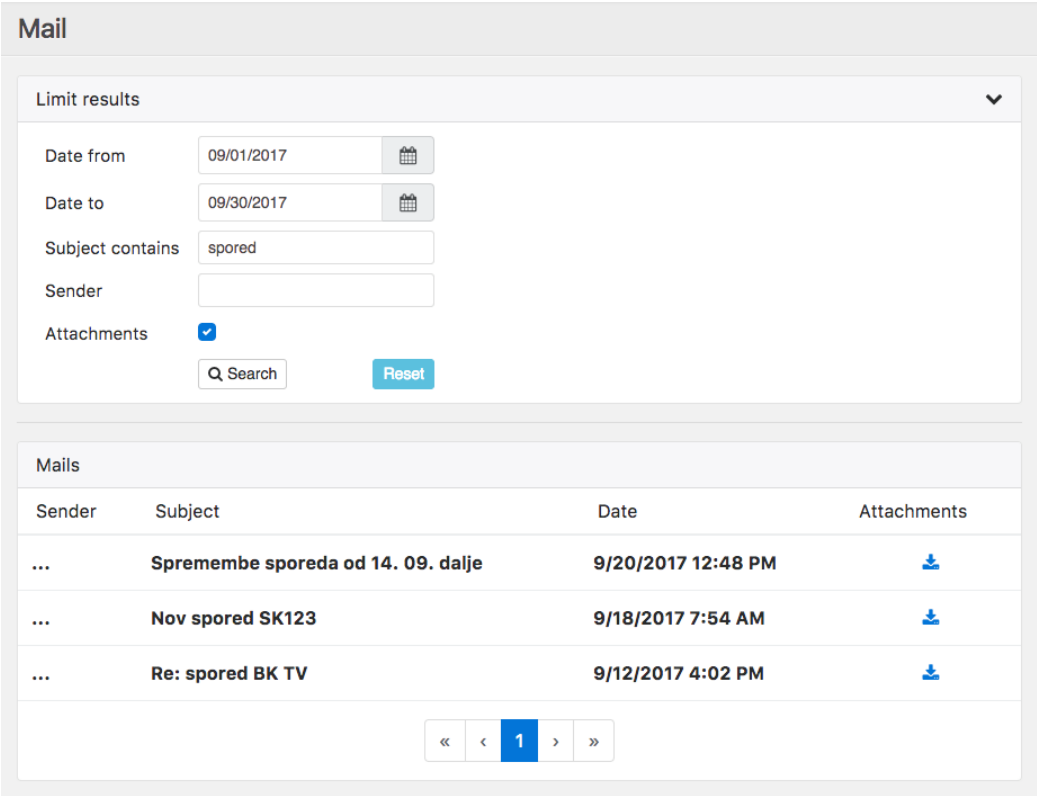
pred potrditvijo najprej hitro pregleda.

V tretjem predelu urednik izbere kam želi shraniti oddajo. Na voljo ima dve možnosti – v zbirko oddaj kanala ali glavno zbirko oddaj. Razlog za izbiro ene izmed možnosti je v tem, da nekatere TV hiše za svoje kanale zahtevajo drugačne podatke (na primer opis ali slike), kot so prikazani pri drugih. Posledično se mora urednik odločiti ali bo obstoječo oddajo prepisal v glavni zbirki ali pa jo dodal v zbirko kanala. Zatem se mu pokažejo gumbi za shranjevanje oddaje, ki omogočajo ali nadaljevanje na prejšnjo oziroma naslednjo obstoječo oddajo ali na podstran za upravljanje sporeda za kanal. S klikom na izbran gumb se postopek upravljanja oddaje zaključi.

Sklop sporočil je namenjen prebiranju e-pošte s strani virov. Urednik vidi vsa sporočila za kanale, ki jih ima v uredništvu. Celoten sklop sestoji iz ene same podstrani, razdeljene na tri predele. Dva sta prikazana na sliki 4.18. Prvi obsega iskanje sporočil glede na datum, vsebovanje niza v zadevi sporočila, na podlagi pošiljateljevega naslova ali na osnovi tega, če ima sporočilo priponke. Drugi predel prikazuje seznam sporočil, ki ustrezajo izbranim omejitvam prvega predela. Izpostavlja pošiljatelja, zadevo sporočila, datum prejetja in v primeru, da ima ta priponke, še gumb za prenos le-teh v arhivski datoteki. Tretji predel se prikaže ob izbiri sporočila s seznama. V zgornjem delu komponente se izpiše celotna vsebina sporočila, v spodnjem pa so prikazane vse priponke. Urednik lahko posamezno priponko prenese. V večini primerov so to dodatna sredstva ali izvorni dokumenti. S prejeto vsebino konstantno spremlja dogajanje na kanalih, saj sporočila poleg izvornih dokumentov vsebujejo tudi programske spremembe in napovedi. Tako zagotovi, da je spored čimbolj ažuren in vsebuje točne podatke.

4.6 Dodatni izzivi

Pri razvoju spletne aplikacije smo se srečali z dodatnimi izzivi, ki vplivajo na prikaz in delo z uporabniškim vmesnikom. Rešili smo jih z uporabo orodij in



Slika 4.18: Sklop sporočil, v okviru katerega sta prikazana predel z iskanjem in predel s seznamom najdenih sporočil.

funkcij, ki so del Angularja. Opisali bomo štiri najpomembnejše.

4.6.1 Prevajanje kode

Aplikacija se je tekom razvoja širila tako funkcionalno, kot tudi po velikosti. Ločevanje funkcionalnosti v ločene komponente in module se izkaže kot zelo dober pristop. Velik del teh je vnovič uporabljen vsaj enkrat, kar smo za nadzorne gradnike iz podpoglavja 4.4 dokazali v okviru uporabnikove poti v podpoglavju 4.5. Med razvojem smo uporabljali način prevajanja JIT, saj so popravki kode v nekaj trenutkih vidni v odjemalcu. Obratno je z načinom AOT, ki za prevajanje zahteva več časa. Večinoma je to nekaj minut in je zaradi tega bolj primeren za izgradnjo distribuirane različice. Prevajalnik AOT

med delovanjem poskuša optimizirati kodo tako, da jo odjemalec čim hitreje interpretira. Pri večjih aplikacijah se čas optimiziranja podaljša, vendar so rezultati običajno vredni čakanja. Opozoriti je potrebno še, da mora biti koda gradnikov kompatibilna z AOT, saj v nasprotnem primeru prevajanje javi napako. Tekom razvoja smo zaradi tega občasno prevedli aplikacijo z AOT in odpravili morebitne napake. Ko smo zaključili razvoj opisanih sklopov, je velikost aplikacije v načinu JIT znašala 7 MB. Po prevajanju z AOT je drastično upadla na 1,4 MB.

4.6.2 Večjezičnost vmesnika

Ena izmed zahtev pred začetkom razvoja aplikacije je bila podpora več jezikom. Nekateri tuji kanali morajo imeti podatke izpolnjene v primarnih jezikih. Posledično za njih skrbijo uredniki, ki obvladajo določen jezik. Za boljše razumevanje operacij v aplikaciji s strani tujih urednikov, je smiselno, da so predstavljene v njihovem jeziku. Tako se izognemo morebitnim napačnim interpretacijam, hkrati pa je uporabniški vmesnik za uporabnika bolj prijazen.

Podpora za večjezičnost v okviru Angularja je dobra. Samo ogrodje omogoča zamenjavo izvornih nizov s prevedenimi. Prav tako obstaja nekaj knjižnic in pristopov, s katerimi lahko poskrbimo za nalaganje izbranega jezika. Bistveno razliko med orodji ogrodja in knjižnicami, sta v času razvoja aplikacije predstavljala način prevajanja in hitrost prikazovanja nizov. Angular je podpiral večjezičnost v predlogah komponent, medtem ko pridobivanje prevedenih nizov v kodi ni bilo mogoče. Na drugi strani so bile na voljo knjižnice in lastni pristopi k prevajanju, ki odpravljajo ravno to odsotnost. Vprašanje, ki smo si ga zastavili se je nanašalo na to, katero možnost izbrati iz vidika hitrosti.

Izbrali smo integrirano podporo v Angular, saj gre za domorodno podporo večjezičnosti[3]. Ključni razlog je predstavljal postopek prevajanja, manjšo težavo pa nepodprtost prevodom v kodi. Prevode v predlogah komponent

[3] Angular - Internationalization (i18n). URL: `https://angular.io/guide/i18n`. [Dostopano: 28.10.2017]

označimo z atributom `i18n`, ki mu kot vrednost podamo identifikator za označevanje prevoda. Atribut umestimo na značko, ki vsebuje izvorni niz za prevajanje. Primer prikazuje izsek programske kode 4.1.

```
<span menu-title i18n="@@navigationOverview">Overview</span>
```

Izsek programske kode 4.1: Primer uporabe atributa i18n.

Za prevajanje nizov aplikacije je potrebno uporabiti orodje `ng-xi18n`, ki izlušči vse izvorne nize iz predlog in jih zapiše v strukturirano datoteko XML za prevajanje. Nato jo uporabimo v ustreznih orodjih ter prevedemo nize za vsak želen jezik. Primer vozlišča v omenjeni datoteki, ki vsebuje izvorni niz in prevod, prikazuje izsek programske kode 4.2.

```
<trans-unit datatype="html" id="navigationOverview">
  <source>Overview</source>
  <context-group purpose="location">
    <context context-type="sourcefile">
      app/dashboard/navigation/navigation.component.ts
    </context>
    <context context-type="linenumber">3</context>
  </context-group>
  <target>Pregled stanja</target>
</trans-unit>
```

Izsek programske kode 4.2: Vozlišče datoteke XML z identifikatorjem prevoda, izvornim nizom, lokacijo pojavitve niza in prevodom.

Kadar aplikacijo za distribuiranje prevajamo z AOT, moramo to storiti za vsak jezik posebej. Pri tem načinu je v distribucijskem paketu (ang. bundle) vsak izvorni niz dejansko zamenjan s prevedenim nizom. Tako ni potrebe po nalaganju prevodov med izvajanjem aplikacije, vendar so že del predloge. Na koncu imamo aplikacijo v vsakem jeziku v ločenem paketu. Za prikaz pravilnega jezika uporabniku, se je smiselno opreti na zaledje ali neposredno na spletni strežnik. V okviru tega določimo kateri paket bomo nudili uporabniku. Slika 4.19 prikazuje primer prevoda aplikacije v slovenščino.

S takšnim pristopom k podpori večjezičnosti vmesnika si zadamo tudi

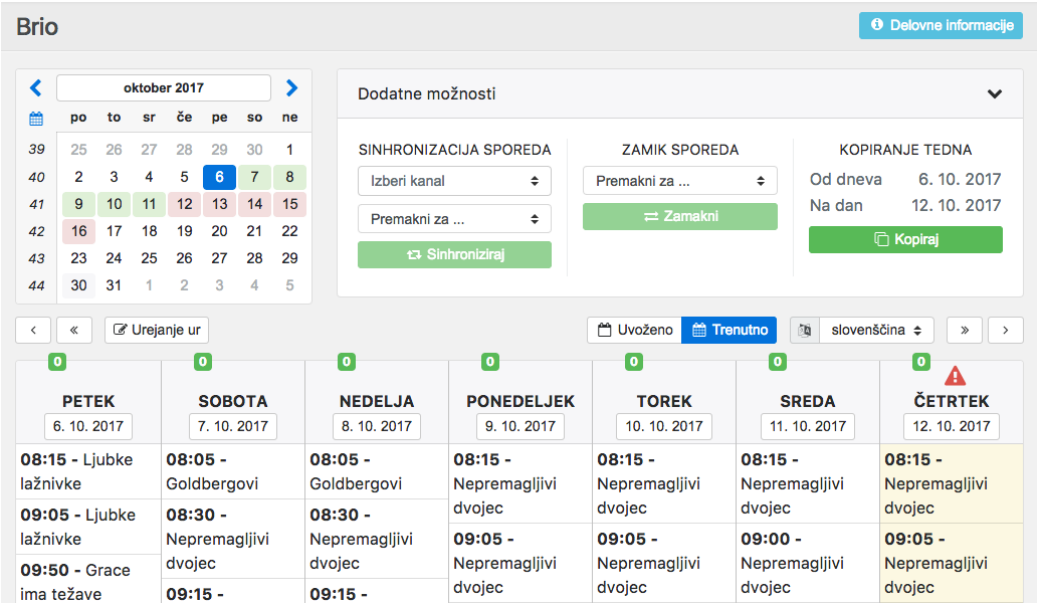
nekaj dodatnega dela. Potrebno je najti način kako prevesti nize v okviru programske kode. Večino nizov se sicer brez težav lahko umesti v predlogo, a ponekod to preprosto ni mogoče. V naši aplikaciji smo na to težavo naleteli zgolj pri prikazovanju obvestil o uspehu operacij, v okviru obvestilne komponente (ang. toast component). Zgradili smo jo tako, da v predlogi v glavno komponento (poimenovana `ng-toaster`) zapišemo podkomponente (poimenovane `ng-toast`). Namesto zapisa niza v programsko kodo, ga umestimo v atribut podkomponente. V programski kodi določimo številčno kodo obvestila, ki se mora izpisati. S storitvijo *ToasterService* (del modula komponente) nato sprožimo dogodek, ki prikaže pravilno obvestilo na podlagi številčne kode. Primer predloge z rešitvijo je prikazan v izseku programske kode 4.3.

```
<ng-toaster>
  <ng-toast [code]="0"
   title="Emission remove"
   i18n-title="@@scheduleMessagesEmRemoveTitle"
   contentProgress="Removing emission ..."
   i18n-contentProgress="@@scheduleMessagesEmRemoveCntProg"
   content="Emission was successfully removed!"
   i18n-content="@@scheduleMessagesEmRemoveCnt"
   contentError="There was a problem with removing emission."
   i18n-contentError="@@scheduleMessagesEmRemoveCntErr">
  </ng-toast>
  <!-- ... Ostala obvestila oziroma podkomponente ... -->
</ng-toaster>
```

Izsek programske kode 4.3: Rešitev problema z izpisovanjem nizov iz programske kode s pomočjo številčnih kod.

4.6.3 Zaznavanje sprememb podatkov

Sistem mora zaradi preprečevanja zlorab pri vnesenih podatkih podpirati zgodovino sprememb. Za vsako spremembo, ki jo uporabnik naredi, mora biti zabeleženo kaj točno je bilo spremenjeno, kdo jo je naredil in kdaj. Pri



Slika 4.19: Prevod podstrani za upravljanje sporeda za kanal v slovenščino.

izbiranju celote sistema, ki bo podpirala zaznavanje, smo določili odjemalski del. Primerjava sprememb v okviru zaledja predstavlja preveč režije, saj zahteva pridobivanje podatkov iz baze, primerjanje in zapis sprememb. Prvi dve operaciji enostavno nadomestimo z odjemalskim delom, hkrati pa ni možnosti zlorabe. V kolikor uporabnik pošlje več sprememb, kot bi jih sicer naredil, lahko kvečjemu povzroči škodo sebi.

Zaznavanje sprememb podatkov se vrši v komponenti obrazca, ki smo jo opisali v podpoglavju 4.2. Hrani glavni objekt z definicijo polj obrazca tipa *FormGroup*. Ta izpostavlja opazovanec `valueChanges`, na katerega se lahko naročimo in pridobimo spremembe na celotnem obrazcu. Za primer vzemimo obrazec s polji, navedenimi v izseku programske kode 4.4. Vključuje tiste, ki smo jih opisali pri podstrani skupin kanalov v podpoglavju 4.5.2. Izsek kode vsebuje lastnost `fb` tipa *FormBuilder*, ki je inicializirana preko vstavljalca odvisnosti. Lastnost `form` je deklarirana s tipom *FormGroup*. Razred `Validators` izpostavlja metode za preverjanje pravilnosti polj. V primeru, da se naročimo na omenjeni opazovanec, bomo ob vsaki spremembi

kateregakoli polja prejeli objekt z vrednostmi vseh.

```
this.form = this.fb.group({
  name: ['', Validators.required],
  description: '',
  channels: ['', Validators.required]
});
```

Izsek programske kode 4.4: Skupina elementov obrazca, ustvarjena s storitvijo *FormBuilder*, ki je del modula *FormsModule*.

Težava takšnega načina zaznavanja je, da ne vemo točno, vrednost katerega polja se je spremenila. Seveda lahko primerjamo celoten prejšnji objekt z novim, a je operacija prezahtevna. Prvotna ideja je bila, da namesto naročnine na skupino polj, združimo (ang. merge) naročnine vsakega izmed njih. Pri tem poleg spremenjene vrednosti, ki jo prejmemo od opazovanca `valueChanges`, podamo še ime polja. To storimo s pomočjo operatorja preslikave (ang. map). Vanj podamo novo vrednost, v našem primeru objekt z obema vrednostima. Po naročanju na združene opazovance nam preostane samo še primerjanje izvorne, pretekle (če obstaja) in trenutne vrednosti po narejeni spremembi. Sliši se kot trivialna operacija, a je njena zahtevnost odvisna od tega kako primerjamo objekte in polja. Odločitev o načinu implementacije bomo prepustili bralcu. Primer opisanega postopka prikazuje izsek programske kode 4.5.

```
const formChanges = Observable.merge(
  ...Object.keys(this.form.controls).map((field: string) =>
    this.form.get(field).valueChanges.map(
      (value: any) => ({ field: field, value: value })
    )
  )
).subscribe(
  (change: IValueChange) => this.detectChange(
    this.originalModel, // Objekt z izvornimi vrednostmi
    this.model, // Objekt s trenutnimi vrednostmi
```

```
    change.field,
    change.value
  )
);
```

Izsek programske kode 4.5: Zaznavanje sprememb posameznega elementa obrazca.

Tak način zaznavanja sprememb je dobro deloval vse dokler nismo pričeli z razvojem podstrani za upravljanje oddaj, opisane v podpoglavju 4.5.3. V sklopih, ki jih upravlja skrbnik, so podstrani z obrazci vsebovale večinoma le polja tipa *FormControl*. Težave so se pojavile pri gnezdenju skupin polj, saj smo ponovno naleteli na že prej opisano težavo. Primerjanje celotnih objektov je zahtevno, le da se je tokrat celoten postopek prestavil za nivo nižje. Končna rešitev, ki odpravlja tudi to težavo je rekurzivno sestavljanje poti do polja. Namesto preproste preslikave sedaj iteriramo čez vse ključe polj in preverimo njihov tip. V kolikor gre za objekt tipa *FormControl*, sestavimo njegovo celotno pot in dodamo spremenjeno vrednost. Če je objekt tipa *FormGroup*, rekurzivno dodamo njegova polja. V okviru primerjave vrednosti je sedaj potrebno razčlenjevanje poti do končnega polja, šele potem sledi dejansko primerjanje. Implementacijo opisanega prepuščamo bralcu.

Z opisanim načinom smo uspešno rešili zaznavanje sprememb podatkov. Zaledju preostaja beleženje prejetih podatkov. Omeniti moramo, da ima ta pri posodabljanju podmnožice podatkov manj dela, sploh pri gnezdenih obrazcih. Težje pride tudi do anomalij, ki se sicer lahko pojavijo pri pošiljanju celotnega nabora podatkov s strani odjemalca.

4.6.4 Izris obrazca na osnovi delno strukturiranih podatkov

Nastavitve v obrazcih, ki so del podstrani sklopov, ki jih upravlja skrbnik lahko postanejo zelo kompleksne. Primer je podstran s kanali, ki smo jo opisali v podpoglavju 4.5.2. Pri dodajanju virov je na voljo več načinov prenosa. Vsak zahteva vnos sebi specifičnih podatkov, na podlagi katerih zaledje po-

skuša dostopati do izvornih dokumentov. V hipu se lahko pojavi potreba po dodajanju novega načina prenosa, ki v času razvoja ni bil predviden. V kolikor se implementacije lotimo tako, da na odjemalski in zaledni strani statično podpremo vse načine, je razvoj zagotovo hitro končan. S tem pristopom ni nič narobe vse dokler ni potrebe po dodajanju novih načinov prenosa. V tem primeru se zna zgoditi, da implementacija zahteva precej več časa, upada pa tudi kvaliteta kode. Namesto neželenega statičnega dodajanja načinov in nepotrebnem večanju odjemalske aplikacije, smo ubrali dinamičen pristop k izrisu takšnih obrazcev.

Najprej smo določili strukturo, ki jo razumeta obe celoti. Osredotočili smo se na njeno generičnost, tako da jo lahko uporabimo za kakršnekoli nastavitve. V glavno plat strukture spadajo ime, tip in dodatne možnosti. Ime je namenjeno prepoznavanju sklopa možnosti s strani uporabnika. Tip je dejanski podatkovni tip, ki je uporabljen na zaledju in je namenjen prepoznavanju vrste možnosti. Dodatne možnosti predstavljajo polja obrazca, ki jih moramo izrisati v vmesniku. Nadaljevali smo z določanjem strukture polja v dodatnih možnostih. Ta sestoji iz oznake, imena, tipa, možnosti o zahtevanju prisotnosti vrednosti in njene oblike, ki jo mora zagotoviti uporabnik. Oznaka je namenjena prepoznavanju možnosti s strani uporabnika. Ime določa ključ, pod katerim mora odjemalec poslati vrednost zaledju. Tip je ponovno dejanski podatkovni tip, le da tokrat ni namenjen le zaledju, temveč na osnovi tega odjemalec ugotovi vrsto polja. Oblika je odvisna od tipa in lahko vključuje regularni izraz za potrjevanje veljavnosti ali pa seznam možnosti, ki jih izbira uporabnik. Krajši primer strukture v formatu JSON, uporabljen za prej omenjene načine prenosa, je prikazan v izseku programske kode 4.6.

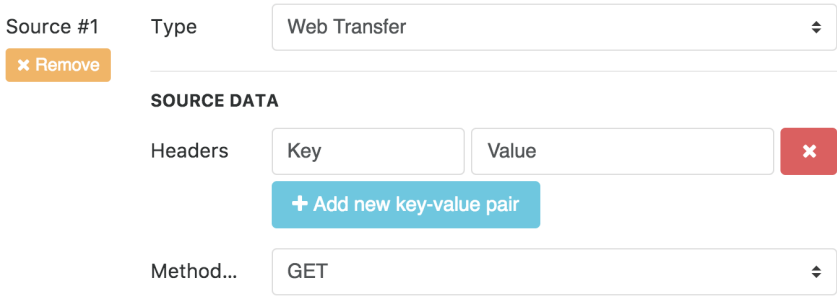
```
{
  "type": "WebTransfer",
  "name": "Web Transfer",
  "options": [{
    "label": "Headers",
    "name": "headers",
```

```
    "type": "multivaluefield[string]",
    "format": null,
    "required": false
  }, {
    "label": "Method Type",
    "name": "methodType",
    "type": "httpmethodtype",
    "format": [{
      "id": 0,
      "text": "GET"
    }, {
      "id": 1,
      "text": "POST"
    }],
    "required": true
  },
  ...
}
```

Izsek programske kode 4.6: Struktura podatkov iz katerih se izriše dinamičen obrazec.

Zatem smo implementirali način tvorjenja strukture na zaledju, nato pa se posvetili razčlenjevanju na odjemalcu. Implementirali smo ga v storitev *SourceOptionsService*, ki pripravi objekt obrazca tipa *FormGroup*. Potrebno je bilo poskrbeti še za prikaz vseh polj obrazca. Izdelali smo glavno komponento *SourceSelectorComponent*, ki po prejemu strukture sprva sproži metodo storitve ter vrne objekt obrazca. Na osnovi strukture nato izriše polja. Izdelali smo ločeni komponenti za prikaz navadnih in strukturiranih vrednosti. Navadne vrednosti zajemajo podatkovne tipe, kot so `string`, `number` in `boolean`. Izriše se vnosno polje, ki ustreza njihovi predstavitvi (glej podpoglavje 4.4). Strukturirane vrednosti ločimo na objekte in polja. Pri objektih se izriše gumb za dodajanje para ključa in vrednosti. Ključ je predstavljen s klasičnim vnosnim poljem, medtem ko je vnosno polje vrednosti ponovno odvisno od podatkovnega tipa – gre za navadno vrednost. Pri poljih je prikaz enak, le da ne vključuje ključa. S tem se celoten proces izrisa dinamičnega

obrazca zaključi. Poudariti je potrebno, da gre za polno funkcionalni obrazec, na katerem prav tako lahko zaznavamo spremembe. Primer izrisa krajše strukture je prikazan na sliki 4.20.



Slika 4.20: Obrazec izrisan na osnovi strukture podatkov iz izseka programske kode 4.6.

Poglavje 5

Namizna aplikacija

Z razvojem spletne aplikacije smo v celoti podprli proces priprave sporedov iz vidika odjemalca. Prednosti, ki jih prinaša sta predvsem dve. Dosegljivost vmesnika iz vseh lokacij, kjer imamo dostop do svetovnega spleta in zmožnost uporabe v brskalniku, ki je neodvisen od platforme. Po proučevanju načina dela urednikov kanalov smo ugotovili, da omenjeni prednosti pripomoreta k bolj učinkovitemu delovnemu procesu, a še vedno obstaja prostor za izboljšave. Zaradi spremljanja sprememb virov morajo na vsake toliko časa odpreti aplikacijo in preveriti, če je v zvezi s sporedom kaj novega. Težava se kaže predvsem v koraku odpiranja aplikacije, zaradi česar smo se odločili, da izdelamo namizno aplikacijo, ki bo to odpravila.

V nadaljevanju diplomske naloge se bomo najprej osredotočili na tehnologijo, ki nam omogoča prenos izkušnje spletne aplikacije na namizje. Nato sledi spoznavanje namena aplikacije ter opis, kako smo se lotili razvoja. Na koncu bomo izpostavili vse implementirane funkcionalnosti in izzive, na katere smo naleteli tekom razvoja.

5.1 Electron

Electron je ogrodje, ki omogoča razvoj namiznih aplikacij s pomočjo spletnih tehnologij. Podpira izvajanje aplikacij na najbolj uporabljenih namiznih plat-

formah – Linux, macOS, Windows. Izpostavlja dve ključni funkcionalnosti s katerima se približamo delovanju namiznih aplikacij. Prva je aplikacijski programski vmesnik, preko katerega dostopamo do zmožnosti domorodne platforme. Druga je namenjena prikazu uporabniškega vmesnika ter posrednemu dostopu do programskega vmesnika. Slednja temelji na brskalniku Chromium in za razliko od klasičnega delovanja omogoča neomejen dostop do platforme na kateri se izvaja. K izvajanju funkcionalnosti, ki so za delovanje odvisne od platforme, pripomore integracija z Node.js. Ogrodje z opisanim pristopom zagotovi možnost razvoja povsem običajne namizne aplikacije, s katerim se približa domorodnim razvojnim tehnologijam končne platforme. Sovpada tudi z idejo enostranskih aplikacij, saj z združitvijo obeh lahko ustvarimo zelo odziven uporabniški vmesnik. Ponuja možnost enostavne implementacije funkcije za samodejno namestitev popravkov. S strani skupnosti so na voljo tudi orodja za pripravo namestitvene datoteke za posamezno platformo. Electron za izvajanje aplikacije uporablja dva osnovna procesa – glavni proces (ang. main process) in izrisovalni proces (ang. renderer process). Za sinhronizacijo obeh omogoča uporabo medprocesne komunikacije (ang. Interprocess Communication – IPC) [2].

5.1.1 Glavni proces

V okviru glavnega procesa se izvajajo vse operacije, ki so neposredno odvisne od zmožnosti platforme. Vanj spada inicializacija vseh gradnikov vmesnika platforme in obravnavanje dogodkov, ki prožijo operacije v ločenih dodatnih procesih ali znotraj izrisovalnega procesa. Del procesa je tudi vhodna datoteka aplikacije, preko katere s pomočjo programskega vmesnika navedemo vse aktivnosti. Te določajo obnašanje gradnikov, vplivajo pa lahko na izvajanje operacij odvisnih od platforme.

5.1.2 Izrisovalni proces

V okviru izrisovalnega procesa se izvaja prikazovanje uporabniškega vmesnika, ki ga pripravimo s spletnimi tehnologijami. Izris opravi brskalnik v katerem so izpostavljene dodatne globalne spremenljivke ali lastnosti, dodane obstoječim objektom, ki so specifične za samo ogrodje. Tako prepoznamo okolje v katerem je prikazan naš vmesnik, hkrati pa dobimo dostop do programskega vmesnika, preko katerega posredno izvedemo operacije odvisne od platforme. S strani uporabniškega vmesnika lahko dostopamo do vseh funkcionalnosti, ki jih sicer omogoča brskalnik. Izrisovalni proces se ustvari za vsako posamezno stran vmesnika, ki jo odpremo poleg že obstoječe [13]. V primeru odprtega primarnega okna aplikacije in nastavitev v ločenem oknu, imamo najmanj dva izrisovalna procesa (dodatni procesi so ustvarjeni v primeru uporabe posebne značke `webview`[1] v okviru posameznega okna).

5.1.3 Medprocesna komunikacija

Z uporabo medprocesne komunikacije poskrbimo za sinhronizacijo stanj operacij med glavnim in izrisovalnim procesom. Temelji na enosmernih imenovanih kanalih, preko katerih pošiljamo poljubne podatke. Znotraj glavnega in izrisovalnega procesa uporabljamo ločena modula `ipcMain` in `ipcRenderer`, ki v vsakem skrbita za pošiljanje in sprejem podatkov. Pred pošiljanjem določimo neko konstantno ime, na osnovi katerega prepoznamo podatke v procesu. Prejemanje podatkov je osnovano na dogodkih, na katere se odzivamo z nastavljenim povratnim klicem.

5.2 Namen uporabe

Namen aplikacije je razširiti možnost uporabe spletne aplikacije (iz poglavja 4) kot samostojne na namizju. Primarno je namenjena urednikom ka-

[1] Electron - <webview> Tag. URL: `https://electron.atom.io/docs/api/webview-tag/`. [Dostopano: 7.11.2017]

nalov. Poleg obstoječih zmožnosti mora podpirati obveščanje uporabnika o spremembah, ki so ključnega pomena v procesu priprave sporeda. V namizno okolje mora integrirati tudi delovno urico, tako da je izmerjen čas dela čimbolj točen.

Namizna aplikacija mora biti uporabniku dosegljiva neprekinjeno, hkrati pa omogočati prikaz vmesnika v izbranem jeziku. Dodatni razlog za razvoj tovrstne aplikacije je v ločevanju spletne aplikacije od običajnega brskalnika iz vidika varnosti in nezmožnosti vpliva njegovih dodatkov na vsebino strani. Pomembno vlogo imajo tudi funkcionalnosti v zadnjih različicah brskalnika, ki je vključen v Electron. Posledično nam ni potrebno podpirati vseh obstoječih brskalnikov, temveč se omejimo zgolj na slednjega.

5.3 Arhitektura in struktura aplikacije

Aplikacija je razdeljena na dve celoti – spletno aplikacijo s podporo namiznim funkcionalnostim in izvajalno okolje (ang. runtime environment), ki skrbi za njeno integracijo z namizjem. Spletna aplikacija arhitekturno ostaja nespremenjena, kar pomeni, da se še vedno izvaja na osnovi platforme brskalnika. Vanjo smo dodali dve storitvi, ki ustrezata vsaka svojemu nivoju predstavitve.

Storitev platforme (ang. Platform Service)

Predstavlja nivo, ki skrbi za zaznavanje in sporočanje vrste platforme na kateri se aplikacija izvaja. Služi kot ovojnica med aplikacijo in funkcionalnostmi, ki jih je možno uporabiti na končni platformi. Za edinstven dostop do storitve, ki implementira proženje funkcionalnosti platforme, izpostavlja lastnost preko katere dostopamo do nje. S tem omogočimo zamenjavo posamezne storitve platforme, brez potrebe po spreminjanju vseh odvisnih gradnikov.

Storitev Electron (ang. Electron Service)

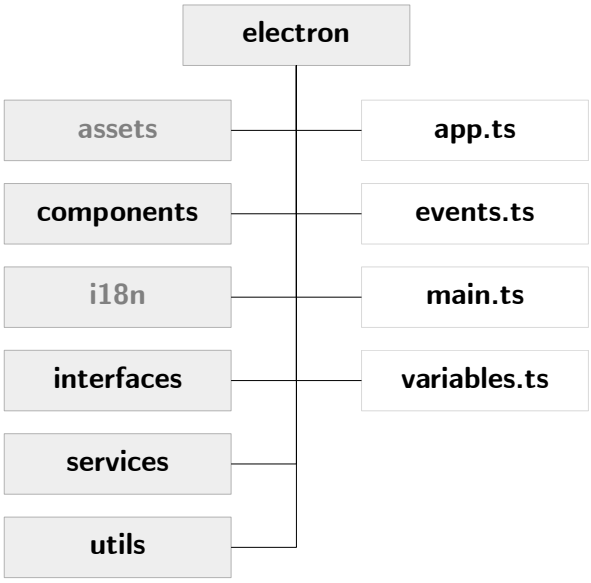
Storitev, ki implementira proženje funkcionalnosti odvisnih od namizne platforme. Predstavlja nivo, ki omogoča komunikacijo s platformo. Vključuje vse možne operacije, ki jih podpira izvajalno okolje. V celoti spletne aplikacije predstavlja vstopno točko za komunikacijo z glavnim procesom in uporablja modul `ipcRenderer`. Za spremljanje dogodkov uporablja poseben predmet, ki omogoča odzivanje na prejete podatke.

Izvajalno okolje je povsem ločeno od spletne aplikacije in je prav tako neodvisno od uporabljenih knjižnic. Namenjeno je izvajanju s strani Electrona, katerega programski vmesnik je uporabljen v kodi. Electron kot ogrodje ne predpisuje nobene strukture, katero naj bi aplikacije upoštevale, temveč o tem odloča razvijalec. Posledično smo se odločili, da bomo ubrali podoben pristop, kot ga uvaja Angular. Posamezne gradnike smo ločili v mape ter v okviru imeniške strukture uvedli naslednje enote:

- **komponente (ang. components)** – vsebuje vse gradnike prikazane v namiznem vmesniku, v okviru vsakega so obravnavane njemu pripadajoče operacije
- **vmesniki (ang. interfaces)** – vključuje infrastrukturno kodo, ki določa pravilno uporabo predpisanih metod ter s tem podpira življenjske cikle aplikacije
- **storitve (ang. services)** – vsebuje vse dogodke, ki jih sprožajo komponente v obliki opazovancev in poskrbi, da je odzivanje nanje v primernih gradnikih
- **orodja (ang. utils)** – vključuje vse logične operacije, ki niso odvisne od gradnikov vmesnika, temveč le izrabljajo funkcionalnosti odvisne od platforme

Poleg naštetih enot se v imeniku nahajata še mapa s sredstvi (vsebuje

datoteke ikon) in mapa s prevodi. Vključene so tudi datoteke, ki so nujne za zagon in delovanje aplikacije. Imeniška struktura je prikazana na sliki 5.1.



Slika 5.1: Imeniška struktura namizne aplikacije. Na levi strani so prikazane enote in pomožne mape (osivljene), na desni pa datoteke namenjene delovanju aplikacije.

V aplikacijsko strukturo okolja smo uvedli tehniko vstavljanja odvisnosti, ki deluje podobno kot v spletni aplikaciji. Uporabili smo jo z namenom, da približamo celotno aplikacijo miselnosti Angularja. S tem smo omogočili, da vhodna datoteka `main.ts`, ki jo izvaja Electron, predstavlja najmanjši modul, ki ne vključuje logike. Namesto tega zgolj kliče obstoječo storitev `app.ts`, ki sproži funkcije potrebne za zagon aplikacije. Tako smo ločili proces inicializacije izven vhodne datoteke in ga lahko izvedemo ob kateremkoli drugem dogodku. Pomembno vlogo pri namiznih aplikacijah igra tudi shranjevanje podatkov, ki se jih v času izvajanja potrebuje za pravilno obnašanje vmesnika. Dodali smo storitev podatkovne shrambe (ang. Data Store Service), ki hrani vse podatke potrebne za delovanje aplikacije. Ti so posledično ločeni od storitev, ki nad njimi izvajajo operacije, hkrati pa do njih lahko

dostopamo od koderkoli.

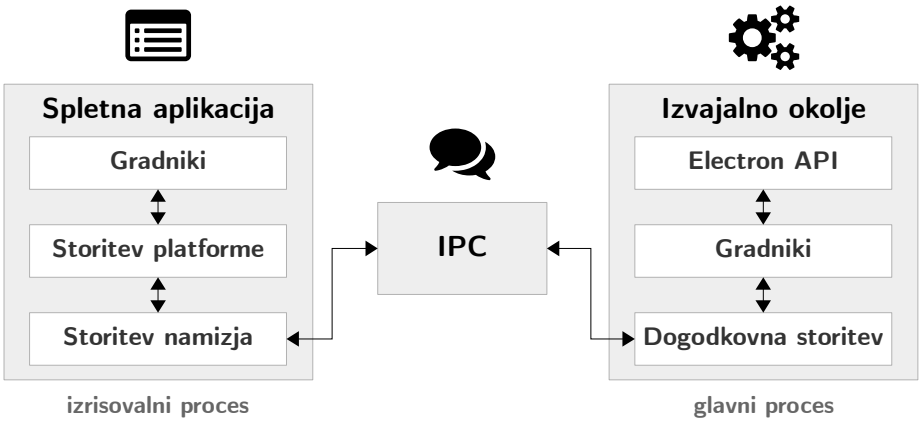
Sporazumevanje med celotama aplikacije je ključnega pomena, saj morata delovati skladno. V ta namen smo pripravili konfiguracijsko datoteko `variables.ts`, ki poleg drugih vključuje spremenljivke s poimenovanjem kanalov za komunikacijo. Uporabljene so kot argument v okviru pošiljanja ali prejemanja podatkov pri medprocesni komunikaciji. V aplikaciji smo definirali dva kanala – enega za glavni proces in enega za izrisovalni proces. Za optimalno uporabo kanalov smo definirali naštevni tip (ang. enumeration type) in razred za dogodke. Shranjena sta v datoteki `events.ts`, ki je prikazana v izseku programske kode 5.1. Pri inicializaciji objekta omenjenega razreda določimo vrsto dogodka ter mu podamo morebitno vrednost. Služi kot ovojnica za pošiljanje podatkov preko IPC, na podlagi katerih enostavno ugotovimo kateri funkciji pripadajo. Na tem mestu naj opozorimo, da IPC podpira zgolj klasične podatkovne tipe JavaScripta, zato se podatek o tipu objekta med prenosom izgubi. To je tudi razlog, da smo se namesto razširjanja osnovnega razreda odločili za uporabo naštevnega tipa.

```
export enum ElectronEventType {
  IDLE_TIME,
  TRAY_CLOCK,
  LANGUAGE,
  NOTIFICATIONS,
  ROUTE,
  LOGGED_IN
}

export class ElectronEvent {
  constructor(
    public type: ElectronEventType,
    public value?: any
  ) {}
}
```

Izsek programske kode 5.1: Definicija naštevnega tipa in razreda, ki sta uporabljena za prepoznavanje dogodkov in prenos podatkov pri IPC.

Konfiguracijska datoteka in datoteka z definicijami sta v spletni aplikaciji uvoženi (ang. imported) v storitev Electron. V okviru izvajalnega okolja sta vsebovani v dogodkovni storitvi *EventHandlerService* ter dogodkovnem orodju *EventHandlerUtil*, ki skrbita za pošiljanje in prejemanje dogodkov. Mapa s celotno imeniško strukturo izvajalnega okolja je posledično vsebovana v istem imeniku, kot mapa s strukturo spletne aplikacije. S tem spletni aplikaciji omogočimo enostaven dostop do obeh datotek. Kljub temu sta celoti še vedno neodvisni, kar pomeni, da spletno aplikacijo lahko uporabljamo samostojno. Slika 5.2 prikazuje način komunikacije med celotama.



Slika 5.2: Arhitektura namizne aplikacije, ki zajema spletno aplikacijo in izvajalno okolje.

5.4 Izbira dodatnih knjižnic

Podpora razširitvam JavaScripta in razvoj nekaterih funkcionalnosti namizne aplikacije sta terjali vključevanje dodatnih knjižnic. Opredelili jih bomo v razširitvene knjižnice, razvojna orodja in domorodne module. Razširitvene knjižnice so namenjene dodajanju funkcionalnosti v programski jezik, s katerimi lahko na boljši ali lažji način zapišemo kodo. Razvojna orodja so namenjena poenostavitvi razvoja in jih v distribuirano različico ni potrebno

vključiti. Domorodni moduli se uporabljajo za razvoj funkcionalnosti na nivoju platforme, kar omogoča neposredno izrabo njenih zmožnosti.

5.4.1 Razširitvene in razvojne knjižnice

Najprej smo dodali že opisano knjižnico RxJS (glej podpoglavje 3.5), s katero smo pridobili zmožnosti za odzivanje na dogodke s pomočjo opazovancev. Vključili smo jo predvsem zaradi boljše strukturiranosti kode in preglednosti operacij. Za podporo vstavljanju odvisnosti smo izbrali knjižnico TypeDI, ki se jo uporablja na podoben način, kot vstavljanje odvisnosti v Angularju. Knjižnica razen dodatne podpore za uporabo metapodatkov v TypeScriptu, ne zahteva drugih odvisnosti. Za podporo izvajanju nalog v povezavi z izgradnjo aplikacije smo dodali orodje Gulp.js. Poleg tega smo dodali še nekaj dodatnih razvojnih knjižnic, ki se opirajo nanj in omogočajo izvedbo zahtevnejših operacij. Namestili smo še knjižnice namenjene poenostavitvi razvoja in razhroščevanju aplikacij z Electronom, ki jih bomo omenili v nadaljevanju.

5.4.2 Domorodni moduli

Pred opisom uporabljenega modula naj pojasnimo, kaj pravzaprav domorodni moduli sploh omogočajo. Electron za izvajanje JavaScripta izven izrisovalnega procesa uporablja Node.js. Posledično je možno uporabiti vse funkcionalnosti, ki jih ta omogoča v okviru namizne aplikacije. Med te spadajo tudi domorodni moduli. Z njimi lahko ustvarimo povezavo med knjižnicami napisanimi v splošnonamenskih programskih jezikih, kot sta C in C++, in JavaScriptom. Načeloma je prevajanje teh potrebno izvesti na končni platformi, s čimer pridobimo možnost dostopa do njenih sistemskih funkcij. Tako se razsežnosti uporabe JavaScripta bistveno povečajo, hkrati pa integracija z namiznim okoljem postane bolj smiselna.

Pri razvoju namizne aplikacije smo se oprli na modul system-idle-time[2].

[2] GitHub - paulcbetts/node-system-idle-time. URL: `https://github.com/paulcbetts/node-system-idle-time`. [Dostopano: 3.11.2017]

Izpostavlja funkcijo, s katero lahko ugotovimo čas neaktivnosti uporabnika na namizju. Pripravljen je za vse tri podprte platforme, kar pomeni, da pri vsaki uporablja njen API. Posamezne funkcije implementirane za vsako platformo nato poveže s funkcijami dosegljivimi iz JavaScripta. Izvršljiva datoteka se pripravi v času namestitve odvisnosti, s čimer je pogojena tudi platforma za katero ta deluje.

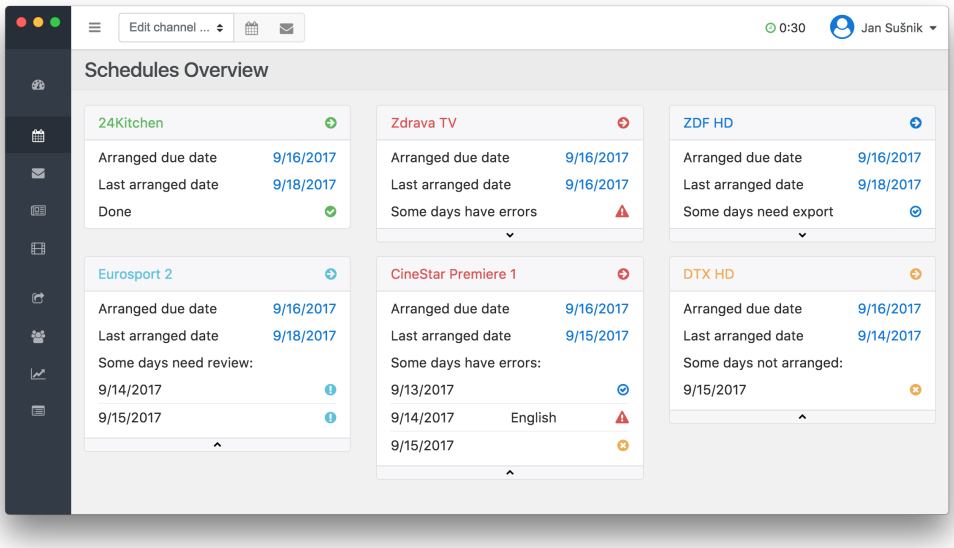
5.5 Uporabnikova pot

Aplikacija v namizno okolje uvaja dodatne funkcionalnosti, ki se osredotočajo na delo urednika kanalov. Glede na to, da ima to vlogo posredno določeno tudi skrbnik, je aplikacija namenjena še njemu. V nadaljevanju bomo opisali prednosti, ki jih namizna aplikacija prinaša uporabnikom in dodali nekaj tehničnih podrobnosti.

5.5.1 Okno vmesnika

Izhodiščno točko za pričetek uporabe aplikacije predstavlja glavno okno z vmesnikom. V njem teče izrisovalni proces, ki prikazuje spletno aplikacijo. Delovni proces je za uporabnika nespremenjen, kar pomeni da se mora sprva overiti. Na osnovi njegovih podatkov nato izvajalno okolje prilagodi jezik vmesnika, omogočijo pa se mu tudi vse zmožnosti namizne aplikacije. Z delom lahko nadaljuje po ustaljeni poti. Za hitrejši dostop in izvajanje prikritih funkcionalnosti se ob zaprtju okna le-to zgolj skrije, še vedno pa deluje v ozadju. Popolni izhod iz aplikacije je možen preko menija sistemske ali orodne vrstice. Posebnost, ki jo Electron na nekaterih platformah omogoča je skrivanje sistemske okrasitve okna. S tem obliko aplikacije lahko prilagodimo za čimboljšo izrabo delovnega prostora. V spletni aplikaciji preko storitve platforme zaznamo platformo na kateri teče in izgled dodatno uravnamo. V naši aplikaciji smo to funkcionalnost izrabili pri različici namenjeni operacijskemu sistemu macOS. Velikost navigacijskega predela se ob pomanjšanju prilagodi širini ukaznih gumbov okna (ang. titlebar buttons), glava strani

pa poleg ostalih možnosti omogoča pomikanje okna po namizju. Electron v okno vključuje tudi menijsko vrstico, ki jo lahko prilagodimo. Osnovne funkcije, kot so razveljavljanje in uveljavljanje sprememb, kopiranje in lepljenje, povečevanje in zmanjšanje velikosti vsebine ter vstop v celozaslonski način, so že podprte. Slika 5.3 prikazuje izgled okna vmesnika na macOS.



Slika 5.3: Okno vmesnika s pomanjšanim in prilagojenim navigacijskim predelom v okviru spletne aplikacije. Prikazana je podstran za pregled sporedov v uredništvu.

5.5.2 Ikona in meni sistemske vrstice

Prikaz stanja aplikacije je viden preko ikone in menija sistemske vrstice (ang. system tray). Stanje določa delovna urica, ki je lahko vključena ali izključena, odvisno od vrednosti v storitvi podatkovne shrambe. Delovna urica je namenjena beleženju časa, ki ga uporabnik porabi za celoten proces priprave sporedov. Hkrati služi kot osnova za obračunski sklop in pomembno prispeva k statistiki, ki obravnava čas namenjen pripravi sporeda. Uporabniku

se urica vključi takoj po prijavi v sistem, izključi pa na podlagi določenih pogojev (opisano v nadaljevanju) oziroma ročno s strani uporabnika. Ikona stanje delovne urice odraža z odtenkom barvnega polnila (Slika 5.4).

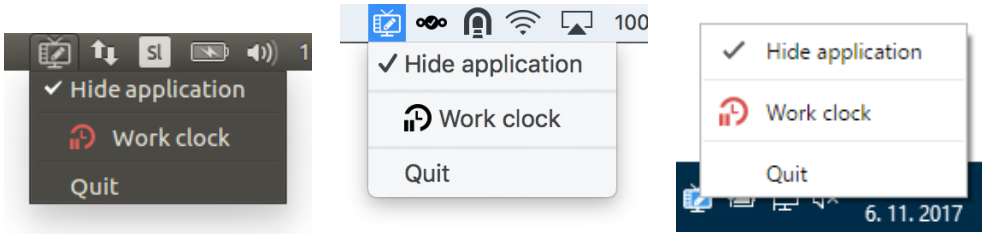
Izključena Vključena

Slika 5.4: Ikona aplikacije v sistemski vrstici, ki je upodobljena glede na stanje delovne urice. Prikazana je za vse tri platforme po vrsti – Linux, macOS, Windows.

V menijski vrstici so na voljo tri možnosti. Prva je namenjena skrivanju ali prikazovanju okna in služi kot bližnjica, kadar okna ni v bližini. Druga predstavlja stanje stikala urice, ki ga je s klikom mogoče vključiti ali izključiti. Tretja možnost je namenjena izhodu iz aplikacije in jo popolnoma zaustavi. Izgled ikone in menija v sistemski vrstici je za vsako podprto platformo prikazan na sliki 5.5. Posebnost programskega vmesnika Electrona je, da posamezne menijske možnosti ne moremo spreminjati po tem, ko smo že določili njen izgled. Običajno to predstavlja manjši problem, ko želimo omogočiti kakšno izmed možnosti ali spremeniti njeno ikono. Rešitev težave predstavlja ponovna izgradnja celotnega menija, kar je pri dogodkovno gnanem pristopu le trivialni klic metode. V našem primeru smo to enostavno rešili z združevanjem vseh opazovancev, ki vplivajo na spremembe menija.

5.5.3 Meni orodne vrstice

Dodatek k možnostim menija sistemske vrstice, so možnosti v meniju orodne vrstice. Prinaša dodatne gumbe, ki so namenjeni preverjanju sprememb in dostopu do podstrani v aplikaciji. Prikaz možnosti je na voljo zgolj na operacijskih sistemih macOS v okviru pristanišča (ang. dock) in Windows v okviru orodne vrstice nalog (ang. taskbar). Dodajanje možnosti na Linuxu Electron ne podpira. Posledično smo za to platformo izpustili tudi ročno



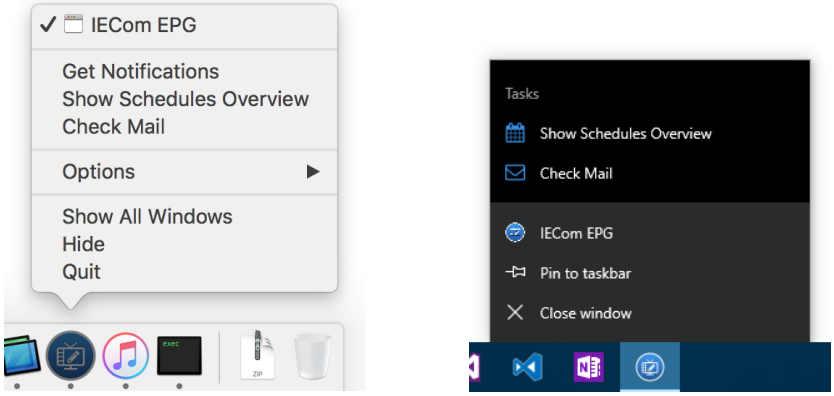
Linux macOS Windows

Slika 5.5: Meni v sistemski vrstici, ki se odpre s klikom na ikono.

dodajanje ob namestitvi aplikacije, saj ni edinstvenega načina za definiranje možnosti na vseh namiznih okoljih. Meni pristanišča na macOS vključuje tri možnosti. Prva omogoča pridobivanje obvestil o spremembah sporeda in novo prispelih e-poštnih sporočilih. Druga prikaže aplikacijo in odpre podstran za pregled sporedov v uredništvu. Tretja možnost prikaže aplikacijo in odpre podstran sklopa sporočil. Vse možnosti glede na programski vmesnik Electrona delujejo na isti način, kot možnosti v meniju sistemske vrstice. Ob kliku na posamezno se sproži klic funkcije, ki izvrši neko operacijo. Drugačen pristop uvaja programski vmesnik za odziv na izbiro v orodni vrstici nalog na Windowsih. Vsaka možnost privzeto kliče izvršljivo datoteko aplikacije, kateri pripne argument. Razlikuje se tudi namembnost nalog v orodni vrstici, saj so primarno namenjene izvedbi ob prvem odpiranju aplikacije. Iz tega razloga smo vanjo dodali le obe možnosti za odpiranje aplikacije in prikazovanje podstrani. Slika 5.6 prikazuje izgled obeh menijev.

5.5.4 Prikrite funkcionalnosti

Spletna aplikacija predstavlja jedro namizne aplikacije, saj je slednja povsem odvisna od nje. Ravno zaradi tega implementira vse ključne funkcionalnosti, s čimer omogočimo neodvisnost od platforme. To pomeni, da v osnovi deluje v privzetem načinu, na posamezni platformi pa lahko prilagojeno. V ozadju spletne aplikacije se izvajata opazovanca, ki skrbita za beleženje delovnega



macOS Windows

Slika 5.6: Meni orodne vrstice, ki prikazuje možnosti za hiter dostop do vsebine.

časa in stalno preverjanje sprememb v sistemu. Prav to je tudi razlog, da se aplikacija ob zaprtju okna zgolj skrije, saj v tem primeru izrisovalni proces še naprej skrbi za izvajanje JavaScripta. Preverjanje sprememb se vrši preko programskega vmesnika REST, kot vsi ostali klici, ki jih opravimo v spletni aplikaciji.

Delovna urica funkcionira na osnovi predmeta, opazovancev in naročnin. Njena funkcionalnost je zajeta v storitvi *WorkClockService*. Privzeto delovanje se prične s klicem metode, ki sproži preverjanje neaktivnosti uporabnika na osnovi dogodkov DOM. Ta se izogiba izvajanju preko izrisovalnika, ker ne podpira obvladovanja dogodkov (preveč specifična funkcionalnost, ki jo je bolje uporabiti neposredno). Začetni dogodek, ki ga prične prazen opazovanec nadomešča dejanske aktivnosti, ki jih v času določenem za neaktivnost lahko sploh ni. V primeru, da uporabnik v petih minutah od zadnje aktivnosti ne sproži kakšnega izmed dogodkov, se delovna urica izklopi. Koda, ki implementira opisano je prikazana v izseku programske kode 5.2. V njej se nahaja polje `events` z imeni dogodkov, ki se smatrajo kot aktivnost. Vključena je

tudi lastnost `idleTime` s časom neaktivnosti podanim v milisekundah. Uporabljena je še metoda `stop()`, ki prekliče vse naročnine in s tem zaustavi beleženje.

```
this.watchSubscription = Observable.merge(
  Observable.of(null),
  ...this.events.map(
    event => Observable.fromEvent(document, event)
  )
)
.switchMap(() => Observable.timer(this.idleTime).take(1))
.subscribe(() => this.stop());
```

Izsek programske kode 5.2: Preverjanje neaktivnosti uporabnika na osnovi dogodkov DOM.

Nadalje se izvede naročnina na opazovanec urice, ki ob 30-sekundnih presledkih sinhronizira stanje z zaledjem. Delovanje se v primeru namizne aplikacije nekoliko prilagodi. Preverjanje neaktivnosti uporabnika je izpuščeno, saj nam to funkcionalnost na nivoju platforme zagotavlja operacijski sistem. Hkrati je veliko bolj zanesljiva, saj uporabnik še vedno lahko opravlja delo izven aplikacije. V okviru tega smo uporabili domorodni modul system-idle-time. Tako se delovna urica ne izklopi na osnovi dogodkov DOM. Spletna aplikacija namesto sinhronizacije stanja preko IPC pošlje zahtevo o sporočanju časa neaktivnosti na namizju. Izvajalno okolje se z uporabo metode modula odzove. Koda, ki je namenjena odgovoru na zahtevo je zajeta v orodju *WorkClockUtil* ter prikazana v izseku programske kode 5.3. Orodje vključuje lastnosti `systemIdleTime` in `eventHandlerService`. Prva kaže na objekt modula, druga pa na storitev za obvladovanje dogodkov. Obe sta inicializirani preko vstavljalca odvisnosti.

```
get idleTime(): number {
  return this.systemIdleTime.getIdleTime();
}

sendIdleTime(): void {
```

```
  this.eventHandlerService.send(
    new ElectronEvent(
      ElectronEventType.IDLE_TIME,
      this.idleTime
    )
  );
}
```

Izsek programske kode 5.3: Metoda za vračanje vrednosti pridobljene s strani modula in metoda, ki pošlje dogodek o času neaktivnosti preko IPC. Uporabljeni sta v okviru izvajalnega okolja.

Po prejemu odgovora na strani spletne aplikacije preverimo, če čas neaktivnosti presega dvominutno omejitev. Pomagamo si z operatorjem omejevanja (ang. filter), ki ob preseganju časa neaktivnosti sproži izklop delovne urice. Preverjanje opravimo enostavno preko opazovanca, ki ga izpostavlja storitev namizja v okviru storitve platforme. Naročnina nanj je prikazana v izseku programske kode 5.4. Storitev namizja vključuje lastnost `idleTimeDesktop`, ki določa čas neaktivnosti na namizju v milisekundah. Sprememba stanja urice se prav tako pošlje izvajalnemu okolju, saj jo ta hrani v storitvi podatkovne shrambe.

```
this.platformService.desktop.idleTime
  .filter(time => time >= this.idleTimeDesktop)
  .subscribe(() => this.stop());
```

Izsek programske kode 5.4: Omejitev opazovanca glede na čas neaktivnosti in zaustavitev delovne urice ob preseganju le-tega.

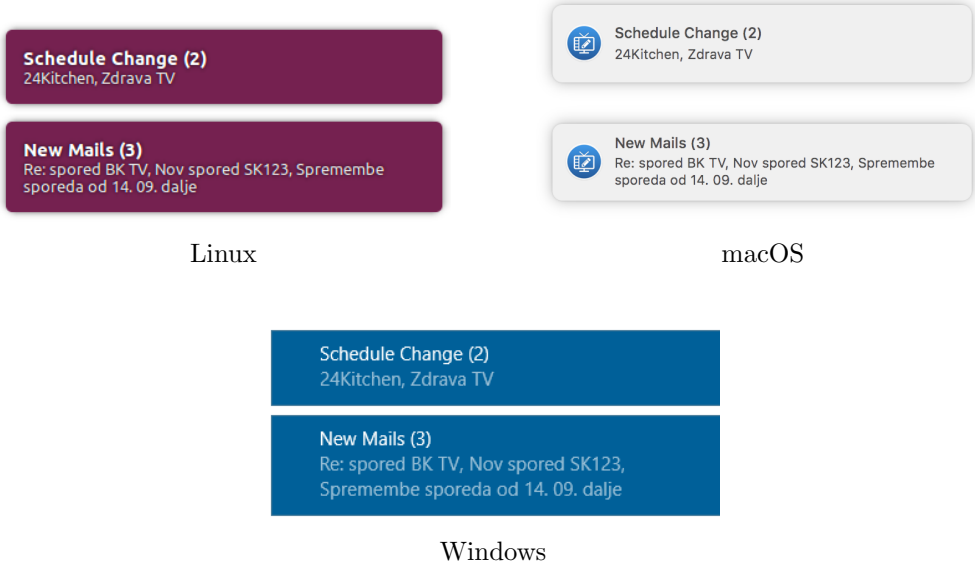
Preverjanje sprememb v sistemu je namenjeno pridobivanju stanja o novih e-poštnih sporočilih in spremembah, ki so bile narejene na kanalih v uredništvu uporabnika. Implementirano je v okviru storitve *NotificationsService*, ki za svoje funkcioniranje prav tako uporablja predmete, opazovance in naročnino. Zajema zgolj privzeti način delovanja, saj spletna aplikacija ne zahteva komunikacije z izvajalnim okoljem. Prične se s proženjem metode, ki v okviru predmeta `checkNotificationsSource` pošlje zahtevo za osvežitev

sprememb. Na njegovem opazovancu se nastavi odštevalnik, ki se sproži takoj in po preteku 5 minut (čas določa lastnost `refreshInterval`). V kolikor se medtem pojavi nova zahteva, se odštevalnik ponastavi in odštevati prične znova. Ob vsakem proženju odštevalnika spletna aplikacija preko metode `fetchChanges` pošlje zaledju zahtevo za pridobivanje nastalih sprememb. Doslej opisan postopek ustreza operaciji izpraševanja (ang. polling). Po prejemu nastalih sprememb se v metodi `compareNotificationsSame` izvede primerjanje z obstoječimi, saj se lahko zgodi, da jih uporabnik še ni opazil in ga zato ne želimo še enkrat opozoriti nanje. V nasprotnem primeru se izvede razpošiljanje obvestil preko metode `executeNotifying`, ki poskrbi, da ustrezne naročnine prikažejo obvestila. Opisan postopek je prikazan v izseku programske kode 5.5.

```
this.checkNotificationsSub = this.checkNotificationsSource
  .switchMap(() => Observable.timer(0, this.refreshInterval))
  .mergeMap(() => this.fetchChanges())
  .distinctUntilChanged(this.compareNotificationsSame)
  .subscribe(
    notifications => this.executeNotifying(notifications)
  );
```

Izsek programske kode 5.5: Naročnina na spremembe, za katere se izvede preverjanje s pomočjo predmeta, opazovanca in dodatnih operatorjev.

V kolikor ne želimo več prejemati sprememb enostavno prekličemo naročnino zajeto v objektu `checkNotificationsSub`. Za obvestila, ki jih prikažemo uporabniku, je v komponenti *NotificationsComponent* vsebovana naročnina, čakajoča na nove spremembe. Nanje se odziva s pomočjo obvestil, ki jih prinaša HTML 5. Posebnost Electrona v tem primeru je, da takšna obvestila samodejno prikaže v okviru platforme. Posledično ni potrebe po medprocesni komunikaciji. Izgled obvestil za vsako končno platformo je prikazan na sliki 5.7. Skupaj s prikazom obvestil se število sprememb prikaže tudi v navigacijskem predelu vmesnika. V kolikor uporabnik prezre obvestilo, ga vmesnik še vedno opozarja glede na sklop, kjer se spremembe odražajo. Slika 5.8 prikazuje primer opozarjanja.



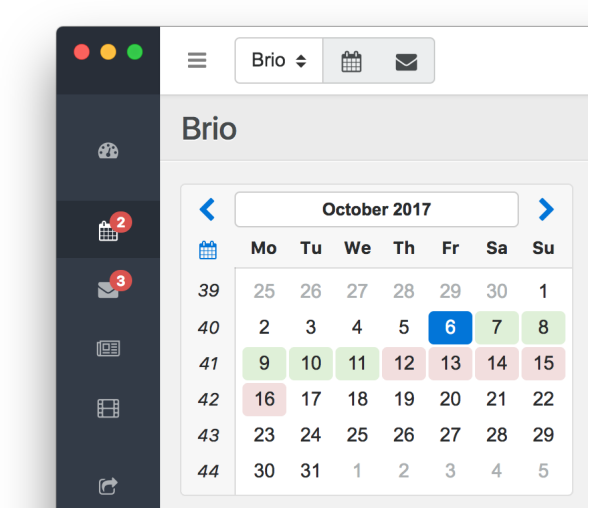
Slika 5.7: Obvestilo o spremembah v sporedih in obvestilo o prejemu novih e-poštnih sporočil.

5.6 Dodatni izzivi

Pri razvoju namizne aplikacije smo dopolnjevali tako spletno aplikacijo, kot izvajalno okolje. Slednje nam je zastavilo dva dodatna izziva, v okviru katerih smo skušali zagotoviti razširitev aplikacije na priljubljene platforme in podporo večjezičnosti.

5.6.1 Prevajanje aplikacije

Priprava aplikacije za izvajanje na končni platformi je zajeta v dva postopka. Prvi obravnava takojšnjo pripravo izvršljive različice, ki je namenjena razhroščevanju. Drugi zajema izgradnjo distribuirane različice, ki jo ponudimo uporabnikom. Oba postopka lahko izvedemo ročno, še bolje pa je, če uporabimo orodje, ki to uredi namesto nas. V našem primeru smo si pomagali z orodjem Gulp.js, ki omogoča razdelitev postopka v delovne naloge (ang. tasks). S tem poenostavimo razumevanje posamezne naloge, hkrati pa



Slika 5.8: Navigacijski predel prikazuje število sprememb v sklopu sporeda in sklopu sporočil.

omogočimo možnost njene ponovne uporabe.

Prva naloga je prevajanje spletne aplikacije v distribuirano različico, kar storimo s pomočjo orodij Angularja. V prvem postopku ta del lahko izpustimo, saj je za namene razhroščevanja bolj smiselno uporabiti način JIT, postavljen na že dosegljivi lokaciji. Sledi naloga prevajanja TypeScripta v JavaScript ter premikanje ustvarjenih datotek v končno mapo. Vanjo v ločeni nalogi skopiramo tudi datoteko `package.json`, ki vsebuje vse odvisnosti na osnovi katerih aplikacija deluje. Prvi postopek na tem mestu vzpostavi povezavo s knjižnico electron-connect, namenjeno ponovnim zagonom aplikacije v načinu razhroščevanja. Zatem se v istem postopku s predpripravljeno različico Electrona naša koda iz končne mape tudi izvede. Hkrati v ozadju deluje prikriti proces, ki preverja spremembe datotek na podlagi katerih se del postopka ponovno zažene. V drugem postopku se namesto tega namestijo odvisnosti na podlagi prej skopirane datoteke z odvisnostmi. Odstrani se tudi nekaj vrstic v JavaScript datotekah, ki so namenjene razhroščevanju. S pripravljenimi datotekami v končni mapi se prične postopek pakiranja, ki je odvisen od izbrane platforme. Zgradi se paket namenjen neposrednemu

izvajanju na ciljni platformi.

Bistvena razlika med obema postopkoma je način pakiranja. Prvi uporablja predpripravljeno različico Electrona, ki je nameščena kot razvojna odvisnost. Posledično se z njo le prične izvajanje prevedene kode, ki je odvisna od vseh nameščenih odvisnosti v projektu. Težava se pojavi, če Electron vsebuje drugačno različico Node.js, kot jo imamo nameščeno na sistemu. Domorodni moduli med različicami običajno niso kompatibilni in jih je potrebno zgraditi za vsako posebej. To poenostavimo s pomočjo knjižnice electron-rebuild, kateri povemo različico Electrona, ta pa nato zgradi domorodne module s pravo različico Node.js. V drugem postopku z orodjem electron-packager zapakiramo izvajalno okolje Electrona skupaj s končno mapo. Ponovno nastane ista težava z različicami, le da tokrat izven projektnega okolja. V končno mapo se namestijo zgolj odvisnosti, ki so nujne za distribuirano različico in jih moramo vnovič zgraditi s pravo različico Node.js. Rešitev predstavlja ločena naloga, ki uporablja funkcijo `rebuild` iz prej omenjene knjižnice za izgradnjo modulov. Za lažje popravljanje delovnih nalog smo pripravili ločen objekt `BUILD_VARIABLES`, ki hrani vrednosti namenjene izgradnji paketa. Primer opisane naloge in uporabe funkcije za izgradnjo modulov je prikazan v izseku programske kode 5.6.

```
gulp.task('build.npm.rebuild', callback => {
  rebuild({
    buildPath: BUILD_VARIABLES.DESKTOP_EXPORT_DIR,
    electronVersion: BUILD_VARIABLES.DESKTOP_ELECTRON_VERSION
  })
  .then(callback)
  .catch(e => {
    console.error('Electron Rebuild Failed');
    console.error(e);
  });
});
```

Izsek programske kode 5.6: Naloga v okviru Gulp.js, namenjena izgradnji domorodnih modulov za pravilno različico Node.js. Uporabljena je za pripravo paketa namizne aplikacije osnovane na ogrodju Electron.

Izgradnja domorodnih modulov se opira na tehnologije, ki so potrebne za prevajanje Node.js. Pri pripravi paketa za izbrano platformo jih moramo naposled imeti nameščene. Velja tudi, da lahko izgradnjo na posamezni platformi opravimo samo za dotično, ne tudi za druge. Nekaj težav pri pripravi modulov smo imeli z operacijskim sistemom Windows, ki pa nam jih je uspelo rešiti s pomočjo orodja windows-build-tools[3]. Za preostali platformi so bile vse zahtevane odvisnosti na voljo v repozitorijih programske opreme in niso povzročale nikakršnih problemov.

Distribuiran paket vsebuje prevedeno aplikacijsko kodo in knjižnice, ki omogočajo prikaz vmesnika ter poganjanje izvajalnega okolja. Velikosti paketov se med platformami ne razlikujejo veliko. Paket za Linux zavzame 143,4 MB, za macOS 131,5 MB in za Windows 151,4 MB. Od tega je velikost aplikacijske kode vključno s spletno aplikacijo 13,5 MB. Upoštevati je potrebno, da večidel kode predstavljajo knjižnice, ki zahtevajo 8,7 MB.

5.6.2 Večjezičnost vmesnika

Jezik namizne aplikacije mora biti konsistenten s spletno aplikacijo. Nastavi se na osnovi podatka, ki ga v vmesniku prejmemo ob prijavi v sistem. Spletna aplikacija izvajalnemu okolju preko IPC sporoči dvočrkovno kodo jezika, na podlagi katere je ta prepoznan. Electron ne ponuja funkcionalnosti za nalaganje prevedenih nizov v izvajalnem okolju, temveč mora to implementirati razvijalec. Glede na rešitve, ki jih skupnost uporablja v okviru Angularja, je s takšnimi pristopi možno podpreti tudi večjezičnost namizne aplikacije.

Odločili smo se, da izdelamo svojo rešitev in ustvarimo storitev *TranslateService*. Deluje na osnovi datoteke JSON, ki vključuje pare ključev in vrednosti, na podlagi katerih naložimo ustrezne prevode. Datoteke se nahajajo v mapi `i18n` in so poimenovane z dvočrkovno kodo jezika. Ob prihodu dogodka o jeziku s strani spletne aplikacije, se kliče funkcija modula `fs` (vključen v Node.js), ki preveri njegov obstoj. V storitvi je privzeti jezik

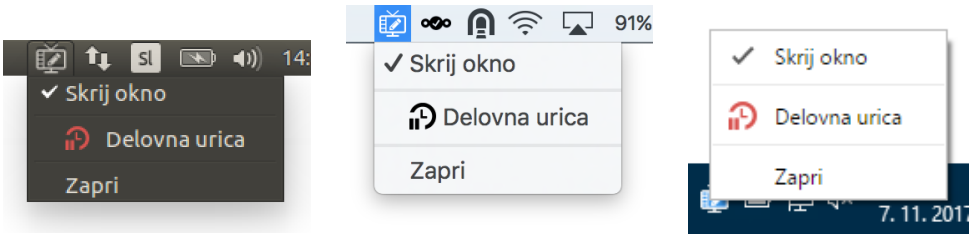
[3]GitHub - felixrieseberg/windows-build-tools. URL: `https://github.com/felixrieseberg/windows-build-tools`. [Dostopano: 5.11.2017]

nastavljen na angleščino, saj obstaja možnost, da želen jezik ni podprt. V kolikor ta obstaja, se nizi preberejo iz datoteke in v obliki objekta shranijo v lastnost `translations`. Storitev nato vključimo v katerikoli drug gradnik in po klicu metode na podlagi ključa (spremenljivka `translation`) pridobimo pravi niz. Podprli smo tudi gnezdenje objektov s prevodi, saj s tem dosežemo boljšo strukturiranost datoteke, ki vključuje prevode. Metoda za pridobivanje prevodov iz objekta je prikazana v izseku programske kode 5.7.

```
get(translation: string): string {
  return translation
    .split('.')
    .reduce(
      (a, b) => a ? a[b] : a,
      this.translations
    );
}
```

Izsek programske kode 5.7: Metoda storitve za prevajanje, s katero pridobimo niz iz objekta s prevodi. Namenjena je prikazu teh v okviru izvajalnega okolja.

Težava, ki jo povzroči tak pristop pridobivanja prevodov se izkaže pri zamenjavi jezika med izvajanjem aplikacije. Takoj ko se jezik zamenja, bi se prevodi morali posodobiti. V našem primeru metoda vrača niz, ki ne zaznava spremembe, lahko pa bi namesto tega vrnili opazovanec. Ta bi posodobil nize takoj po zamenjavi jezika. Opisani način bi sigurno deloval, vendar opazovancev nismo dodali z razlogom. Osveževanje menijev namreč zahteva ponovno uveljavljanje predloge z elementi, preden se ti dejansko posodobijo. Z uvajanjem opazovancev bi postopek zgolj otežili, saj bi morali združiti več opazovancev skupaj (en opazovanec na prevod) in se nato naročiti na glavnega. Namesto tega smo v vseh storitvah raje dodali opazovanec, ki sporoča spremembo jezika in takrat še enkrat uveljavimo predlogo z elementi menija. Slika 5.9 prikazuje slovenski meni sistemske vrstice.



Linux macOS Windows

Slika 5.9: Meni v sistemski vrstici, preveden v slovenščino.

Poglavje 6

Mobilna aplikacija

Podporo procesu priprave sporedov smo s pomočjo razvitih aplikacij zagotovili v predhodnih poglavjih. Preostane nam še proces prikaza, ki se namesto na pripravo podatkov osredotoča na upodobitev teh. Preden smo se lotili izdelave aplikacije, smo razmišljali o izbiri platforme. Podpreti smo želeli dominantna mobilna operacijska sistema v tistem trenutku, a zgolj z eno kodno osnovo (ang. codebase). Odločali smo se med razvojem hibridne, napredne spletne aplikacije in domorodne aplikacije, katere osnova temelji na spletnih tehnologijah. Izbrali smo slednjo, saj po načinu delovanja prekaša učinkovitost in zmožnosti njenih alternativ. Dodatno prednost je predstavljala podpora namenski platformi Angularja, kar omogoči njegovo uporabo izven ustaljenih okvirjev.

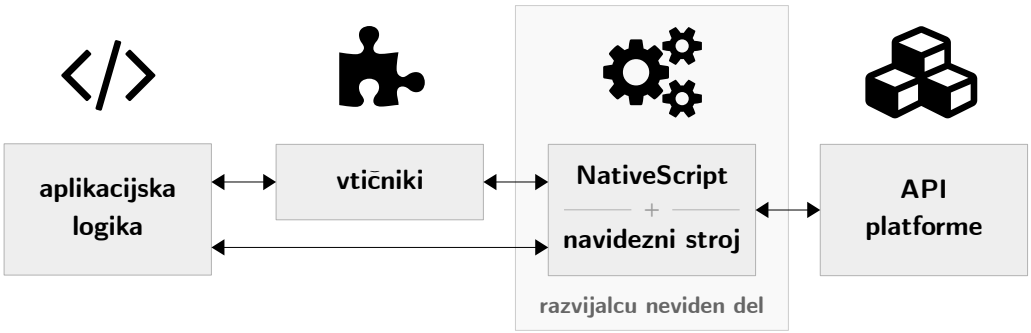
V nadaljevanju bomo najprej opisali tehnologijo, ki omogoča izdelavo domorodnih mobilnih aplikacij z uporabo spletnih tehnologij. Sledi določanje pripadnosti aplikacije glede na vrsto prikaza sporedov in opis njene namembnosti. Ogledali si bomo še arhitekturo in strukturo, ki predstavljata temelj za ustrezno delovanje aplikacije. Opisali bomo, kako so nam pripomogle dodatne knjižnice in predstavili pot za uporabo aplikacije. Za konec sledi pregled dodatnih izzivov s katerimi smo se srečali in jih skušali rešiti med razvojem.

6.1 NativeScript

NativeScript je ogrodje, ki omogoča razvoj mobilnih aplikacij s pomočjo spletnih tehnologij. Deluje kot ovojnica domorodnemu aplikacijskemu programskemu vmesniku, zaradi česar ne uporablja gradnika za prikaz spletne vsebine. Ogrodje definira objekte preko katerih kličemo domorodni API in skrbi za pretvarjanje podatkov iz JavaScripta v obliko domorodnih tehnologij (in obratno). Posledično v najmanjši možni meri vpliva na učinkovitost izvajanja aplikacije. S tem postaja primerljiva izbira za razvoj aplikacije v primerjavi z domorodnimi orodji. Podpira operacijska sistema Android in iOS, za katera omogoča neposreden dostop do njunega APIja [3]. Za razvoj aplikacije z ogrodjem uporabimo enega izmed dveh pristopov – jedro NativeScripta (ang. NativeScript Core) v povezavi s TypeScriptom ali sodobnim JavaScriptom oziroma integracijo NativeScripta z Angularjem (ang. NativeScript with Angular). Bistvena razlika med njima je način razvoja ter uporaba vdelanih programskih vzorcev in principov. Aplikacijo lahko razvijemo bodisi samo za določen operacijski sistem bodisi za oba hkrati. Pri tem delamo zgolj z eno kodno osnovo.

Aplikacije, ki jih izdelamo s pomočjo NativeScripta, delujejo na osnovi navideznih strojev (ang. virtual machines), ki omogočajo prevajanje JavaScripta v strojno kodo [16]. Uporabljeni stroj je odvisen od platforme. Na Androidu je uporabljen V8, ki je prav tako osnova izvajalnega okolja Node.js (omenili smo ga v poglavju 5). iOS uporablja Applov JavaScriptCore. Ogrodje posledično v vsako različico vključi tako našo kodo, kot tudi navidezni stroj. Vse ukaze podane v kodi, nanašajoče se na API platforme, z njegovo pomočjo prevede v ustrezno obliko. Platforma jih tako sprejme kot ukaze iz domorodne aplikacije. To pomeni, da se izvajanje teh v NativeScript aplikaciji zakasni zgolj za čas prevajanja. NativeScript težko uvrstimo v sklop obstoječih vrst spletnih aplikacij (glej podpoglavje 3.1), saj uvaja nekoliko drugačen pristop k razvoju in načinu delovanja. Najbližje je hibridnim aplikacijam, njegovo izvajanje pa je podobno kot pri ogrodju Electron, saj za dostop do domorodnih funkcionalnosti uporablja navidezni stroj.

Jedrne funkcionalnosti NativeScripta lahko razširimo z vtičniki (ang. plugins). Običajno so namenjeni dostopu do zmožnosti operacijskega sistema ali strojne opreme naprave in so odvisni od platforme. Obstajajo tudi vtičniki namenjeni razširitvi funkcij v JavaScriptu in podpori razvojnemu procesu. V kolikor se navezujemo na zmožnosti platforme, moramo funkcionalnost implementirati za vsako posebej. To storimo tako, da jo izdelamo s pomočjo domorodnih orodij, nato pa prenesemo v projekt vtičnika. V njem moramo za vsako platformo izdelati dodatno ovojnico, ki kliče ustrezne metode domorodne implementacije. Namesto lastne izdelave funkcionalnosti, se lahko zanesemo na že obstoječe knjižnice. V ustrezno konfiguracijsko datoteko za platformo vstavimo naslov repozitorija knjižnice in pripravimo ovojnico za klice njenih metod. Za Android so knjižnice vključene s pomočjo orodja Gradle, za iOS pa s pomočjo upravljalnika odvisnosti (ang. dependency manager) CocoaPods. Opisani način delovanja NativeScripta je prikazan na sliki 6.1.



Slika 6.1: Poenostavljen koncept delovanja NativeScripta. Za razvijalca je navidezni stroj neviden – z njim se med razvojem ni potrebno ukvarjati.

Razvoj aplikacij z ogrodjem NativeScript poleg glavnih orodij terja namestitev SDK (ang. Software Development Kit) ciljne platforme. Z njim pripravi datoteko, ki omogoča namestitev in izvajanje aplikacije. Postopek je enak kot pri razvoju z domorodnimi orodji – prevajanje zaženemo vsakič, ko želimo udejanjati spremembe v izvršljivo aplikacijo. Namesto vnovične iz-

gradnje med popravki kode, NativeScript omogoča sinhronizacijo v živo (ang. live sync). S tem se spremenjene datoteke nemudoma posodobijo in ne potrebujejo ponovnega prevajanja. Funkcionalnost je sicer omejena na manjše popravke, vendar kljub temu način razvoja približa spletnim aplikacijam. V kolikor imamo na voljo več naprav, lahko nastavimo, da se spremembe uveljavijo na vseh hkrati (neodvisno od platforme).

6.2 Namen uporabe

Aplikacijo lahko uvrstimo v končni korak procesa prikaza sporedov, ki zajema upodobitev podatkov v uporabniku prijazni obliki. Njen končni uporabnik je gledalec, ki želi spremljati spored TV kanalov na mobilni napravi. Za lažje razumevanje namena aplikacije, moramo najprej definirati njene sklope:

- **spored** – omogoča prikaz oddaj, ki so trenutno na sporedu in prikaz podrobnosti o izbrani oddaji
- **kanali** – vključuje seznam vseh kanalov s sporedom in prikaz oddaj za naslednji dan na posameznem kanalu
- **opomniki** – vsebuje seznam oddaj, za katere mora naprava nekaj časa pred predvajanjem prikazati obvestilo

Aplikacija mora podpirati večjezičnost vmesnika, s tem da ima uporabnik sam možnost izbire jezika. Prav tako mora biti na obeh podprtih platformah implementirana enotna oblika in obnašanje vmesnika. Glavni namen mobilne aplikacije je ozaveščati gledalca o TV sporedu za večje število kanalov. Uvrstimo jo lahko v okvir ostalih elektronskih aplikacij za prikaz sporeda, pri čemer ta ne zagotavlja pretoka za ogled vsebine.

6.3 Arhitektura in struktura aplikacije

Aplikacijo smo izdelali na osnovi NativeScripta integriranega v Angular. S tem smo se približali razvoju enostranske aplikacije, hkrati pa izkoristili pred-

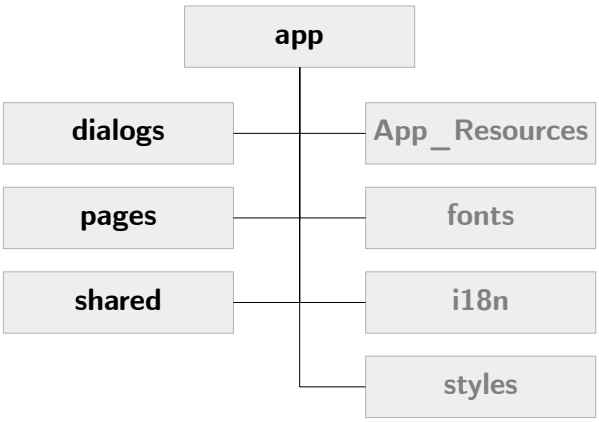
nosti obeh ogrodij. Podatki, potrebni za delovanje aplikacije, so dosegljivi preko REST APIja, ki je ločen od zaledja sistema omenjenega v podpoglavju 4.2. Posledično je za proces prikaza sporedov izdelano svoje zaledje, ki zajame podatke izhoda priprave sporedov. Obdela jih tako, da so v mobilni aplikaciji le prikazani in jih ni potrebno dodatno razčlenjevati.

Posebnost aplikacije je v načinu izrisa, saj namesto brskalniške platforme Angularja uporablja platformo NativeScript. Ta je zadolžena za pretvorbo vseh operacij komponente, ki vplivajo na predlogo. Spremeniti jih mora v tako obliko, da se predloga pravilno izriše v odjemalcu. Namesto označevalnega jezika HTML je uporabljen XML. Z zamenjavo platforme izgubimo tudi dostop do klasičnih globalnih spremenljivk brskalnika, kot sta `window` in `document`. Pridobimo spremenljivke, ki jih v globalni prostor uvaja NativeScript. Ravno v tem je prednost Angularja, saj aplikacije ne razvijamo za brskalnik, ampak za več končnih platform hkrati. V primeru, da bomo poleg mobilne aplikacije želeli imeti še spletno, lahko obstoječi kodi le dodamo ustrezne predloge. Ostale funkcionalnosti imamo že implementirane v logičnem delu komponent. V njem še vedno lahko dostopamo do spremenljivk obeh platform, le da moramo to določiti s pogojem – tak pristop se ne smatra kot najboljša praksa.

Aplikacija navkljub integraciji z dodatnim ogrodjem upošteva strukturne smernice Angularja. Glavne funkcionalnosti so opredeljene v vrhnje enote, prikazane na sliki 6.2, ki se delijo na:

- **dialogi (ang. dialogs)** – zajema manjša okna, ki se uporabniku prikažejo na zahtevo in pripomorejo k opravljanju neke operacije ali prikazu podatkov
- **strani (ang. pages)** – vključuje vse strani aplikacije, po katerih se uporabnik lahko premika
- **skupno (ang. shared)** – vsebuje gradnike namenjene ponovni uporabi v okviru dialogov in strani, nadalje se deli na komponente, razširitve, vmesnike, modele, cevi in storitve, vse skupaj pa zastopa deljeni

modul (ang. shared module)



Slika 6.2: Imeniška struktura mobilne aplikacije. Leva stran predstavlja enote s funkcionalnostmi, desna pa mape namenjene zagotavljanju sredstev aplikacije.

Poleg opisanih enot se v imeniški strukturi nahajajo še dodatne mape, v katerih so sredstva namenjena izgledu (`fonts` in `styles`), prevodom (`i18n`) in nastavitvam specifičnim platformam (`App_Resources`).

Aplikacijska struktura za spremembo od spletne aplikacije ni zastavljena na generičen način. Razlog je v tem, da integracija NativeScripta ne deluje povsem v skladu z načeli enostranskih aplikacij. Bistvo teh je v spreminjanju posameznih predelov strani, kar pa ne ustreza delovanju NativeScripta. Slednji ob vsaki zamenjavi strani ponovno izriše vse gradnike in tako težje ločimo funkcionalne dele. Kljub temu smo dodali tri splošne gradnike, ki predstavljajo zasnovo aplikacije.

Temeljna komponenta

Zajema predmet za obveščanje o uničenju komponente, ki se odzove na pripadajoči dogodek njenega življenjskega cikla. Do njega dostopa komponenta, ki razširja temeljno. Opazovanci, katerih naročnine se morajo z uničenjem

prenehati, imajo kot argument operatorja `takeUntil` podano njeno metodo za pridobivanje opazovanca predmeta. Z njim se prekličejo vse operacije v naročenih opazovancih, kar omogoči sprostitev virov. Služi kot komponenta obliža (ang. patch component), ki dodaja nujno potrebno funkcionalnost na nivo komponente.

Komunikacijska storitev

Predstavlja gradnik aplikacije, ki je zadolžen za pridobivanje podatkov s strani zaledja. Vključuje vse nastavitve in metode, ki so potrebne za pridobivanje podatkov in obvladovanje napak. Služi kot ovojnica nad storitvijo HTTP (ang. Hypertext Transfer Protocol), preko katere poteka prenos podatkov. Prav tako omogoča enostavno zamenjavo protokola, brez potrebe po spreminjanju gradnikov, ki so odvisni od nje.

Komponenta stranske vrstice (ang. Side Drawer Component)

Navigacijski meni s povezavami na strani aplikacije je vsebovan v okviru komponente stranske vrstice. Ta vključuje celotno logiko za njeno odpiranje in zapiranje, prehajanje med stranmi in zadrževanje stanja o trenutno odprti strani. V ostalih komponentah je vključena v predlogi, saj se mora ob spremembi strani ponovno naložiti. Postavitev strani posledično deli na navigacijski del, ki je privzeto skrit in vsebinski del, ki prikazuje dejanske gradnike strani.

Podatkovni tok aplikacije temelji na opazovancih in storitvah. Z njimi vse podatke ločimo v svojo plast, preostalim gradnikom pa izpostavimo zgolj metode za pridobivanje točno določenih. Vse tovrstne storitve, katerih podatki so uporabljeni na več straneh, se kot podenote nahajajo v okviru mape skupno. S pomočjo vstavljanja odvisnosti jim vključimo komunikacijsko storitev `apiService`, preko katere zajamemo zahtevane podatke. Ob klicu APIja za pridobivanje podatkov na opazovancu izrabimo operatorje, ki omogočajo njihovo začasno shranjevanje. Posledično se ob ponovnem klicu ti naložijo

iz predpomnilnika (ang. cache) in prihranijo dodaten zahtevek ter čas nalaganja. V kolikor čas hranjenja podatkov poteče, je ob vnovičnem klicu ponovno zahtevana povezava na API. Primer takšnega opazovanca je prikazan v izseku programske kode 6.1, kjer predmet ustvarjen z operatorjem `publishReplay` hrani podatke eno minuto. Dodaten nivo hrambe podatkov predstavljajo objekti storitev, v katere shranjujemo vse pridobljene podatke. To nam omogoči, da sprva ob klicu manj podrobne končne točke APIja shranimo neko podmnožico podatkov. Ob nadaljnjem odpiranju strani aplikacije tako naložimo že shranjene podatke, vmes pa sprožimo pridobivanje podrobnejših. Iz tega razloga je pomembna tudi temeljna komponenta, saj se v primeru zaprtja strani pred pridobitvijo podatkov zahtevek na API prekliče.

```
this.apiService.get({ path: API_CONFIG.ENDPOINT_CURRENT })
  .publishReplay(1, 60000)
  .refCount()
  .take(1)
  .subscribe((emissions: any) => {
    // Posodobitev pridobljenih podatkov v hrambnih objektih
  }, err => {
    // Obdelava napake in proženje ustrezne operacije
  });
```

Izsek programske kode 6.1: Naročnina na opazovanec, s katerim pridobimo podatke iz APIja in jih za eno minuto hranimo v predpomnilniku.

Podatkovne entitete so zajete v modelih, ki se prav tako v obliki podenot nahajajo v mapi skupno. Implementirane so kot razredi, ki nad vsebovanimi podatki zagotavljajo dodatne metode za poizvedovanje o njihovem stanju. Izdelali smo tri modele, od katerih vsak pripada posameznemu sklopu aplikacije. Za sklop sporeda smo izdelali model *Emission*, ki hrani podatke o oddaji. Poleg tega izpostavlja metode o stanju predvajanja oddaje (se je začela, je končana, se predvaja) in pripadnosti programskemu dnevu (se predvaja danes ali jutri). Sklop kanalov pooseblja model *Channel*, v katerem so shranjeni identifikatorji kanala, ime in ena uporabniško določena spremenljivka. Model *Reminder* je namenjen sklopu opomnikov in hrani identifikator

opomnika, kanal predvajanja ter izvleček o oddaji – identifikator, naslov, čas začetka in konca. Vsi modeli so uporabljeni v okviru podatkovnih storitev njihovih sklopov. Podatki, ki jih prejmemo s strani APIja se pretvorijo v ustrezne objekte kadar je to smiselno. Tako so njihove metode na voljo v delih aplikacije, kjer jih potrebujemo. Kadar želimo le prikazati podatke, se opremo na vmesnik modela. S tem zagotovimo, da podatkov v objekte razredov ne pretvarjamo po nepotrebnem, še vedno pa izrabljamo prednosti, ki jih prinaša definicija objekta.

6.4 Izbira dodatnih knjižnic

Med razvojem aplikacije smo naleteli na nekaj funkcionalnosti, ki so jih že implementirali obstoječi vtičniki oziroma knjižnice. Omejili smo se le na tiste, ki podpirajo obe ciljni platformi in uvajajo enotno obnašanje. Poudariti je potrebno, da jih je v nasprotju z Angularjevimi zbirkami komponent na voljo ogromno. Nekatere izmed teh so ovojnice za priljubljene vtičnike platforme in kot take izpostavljajo samo njihov API. Na drugi strani je možno najti knjižnice, ki v celoti implementirajo funkcionalnost za posamezno platformo. V kolikor knjižnica za želeno funkcionalnost ne obstaja, je to smiselno zapakirati v lastno. S tem odcepimo kodo odvisno od platforme proč od glavne aplikacijske logike.

Uporabili smo tako knjižnice namenjene NativeScriptu in Angularju, kot tudi splošnonamenske. Opredelimo jih lahko na operativne, pomagalne in razvojne. V operativne štejemo tiste, ki v interakciji z uporabnikom izvedejo neko operacijo in so namenjene prikazu vmesnika. Pomagalne so namenjene obdelavi in prikazu podatkov. Razvojne se uporabljajo za podporo in poenostavitev razvoja aplikacije.

Operativne najprej predstavljajo knjižnice, ki so namenjene prikazu vmesnika. Prva je nativescript-theme-core, ki zagotavlja konsistentno obliko vmesnika. Nato smo vključili nativescript-telerik-ui, ki zagotavlja dodatne komponente za interakcijo z uporabnikom – uporabili smo stransko vrstico in pri-

kazni seznam (ang. list view). Dodali smo nativescript-ripple, ki poskrbi za pristno animacijo ob pritisku izbranega elementa. Za okrasitev in prepoznavanje možnosti vmesnika smo dodali nativescript-material-icons, ki vključuje ikone. Sledilo je dodajanje nativescript-ngx-fonticon, ki prikaže znake iz ikonske pisave na podlagi imen razredov CSS. Vključili smo še knjižnici za interakcijo z uporabnikom. Najprej smo dodali nativescript-local-notifications, namenjeno obvladovanju opomnikov. Nato smo vključili nativescript-toast, ki služi prikazu obvestil.

Pomagalne sestavlja knjižnica Moment.js, ki smo jo dodali zaradi podpore obdelovanju datumov in časa. Sprva je nismo vključili zaradi neuporabe vseh funkcij, a se je na iOS z objektom `Date` izkazal problem pri pretvarjanju datumov. Problem bi lahko rešili z dostopom do APIja platforme, vendar smo z vključitvijo knjižnice pridobili še nekaj funkcij, ki so odtehtale sprejeto odločitev. Nato smo dodali nativescript-sqlite, ki služi kot ovojnica z operacijskimi metodami nad implementacijo podatkovne baze SQLite na vsaki platformi. Med pomagalne štejemo tudi ng2-translate, ki zagotavlja podporo za prikaz prevodov.

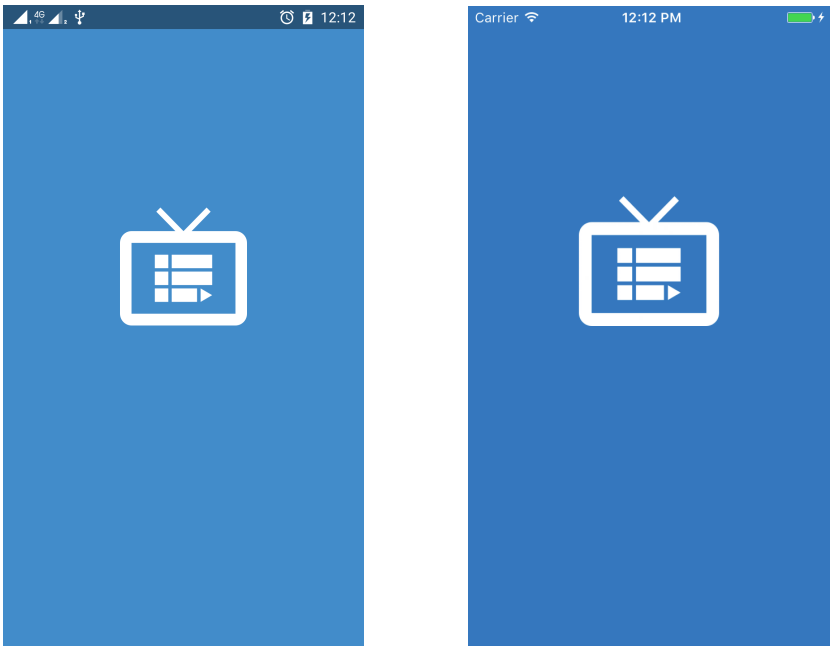
Razvojne knjižnice, ki smo jih uporabili, so namenjene obdelavi oblikovnih predlog in pripravi distribuirane različice aplikacije. Sprva smo vključili knjižnici node-sass in nativescript-dev-sass. Namenjeni sta prevajanju predlog SASS v CSS in izbrisu izvornih oblikovnih predlog iz izvršljive različice. Nato smo vključili še nativescript-dev-webpack, ki služi prevajanju aplikacije v načinu AOT, optimizaciji in pripravi za objavo.

6.5 Uporabnikova pot

Aplikacija je namenjena gledalcem televizije in prikazuje podatke TV sporedov. Pot uporabnika je posledično prilagojena temu, da dotični čimhitreje izve kaj lahko spremlja. V nadaljevanju bomo opisali način uporabe aplikacije, njene lastnosti in nekaj tehničnih podrobnosti.

6.5.1 Uporabniški vmesnik

Uporabnika ob zagonu aplikacije pričaka uvodni zaslon (ang. splash screen) z logotipom aplikacije (Slika 6.3). Namenjen je inicializaciji NativeScripta, tako da lahko aplikacija potem deluje nemoteno. Zatem se uporabniku prikaže prva stran, v okviru katere ima možnost odpreti stransko vrstico. Ta je na voljo zgolj na nekaterih straneh – tam kjer vključimo komponento stranske vrstice, saj v določenih predelih aplikacije namesto tega prikažemo gumb za vračanje na prejšnjo stran. Zgornji del postavitve strani zavzema naslovna vrstica, ki prikazuje njen naslov in dodatne gumbe za izvajanje operacij.

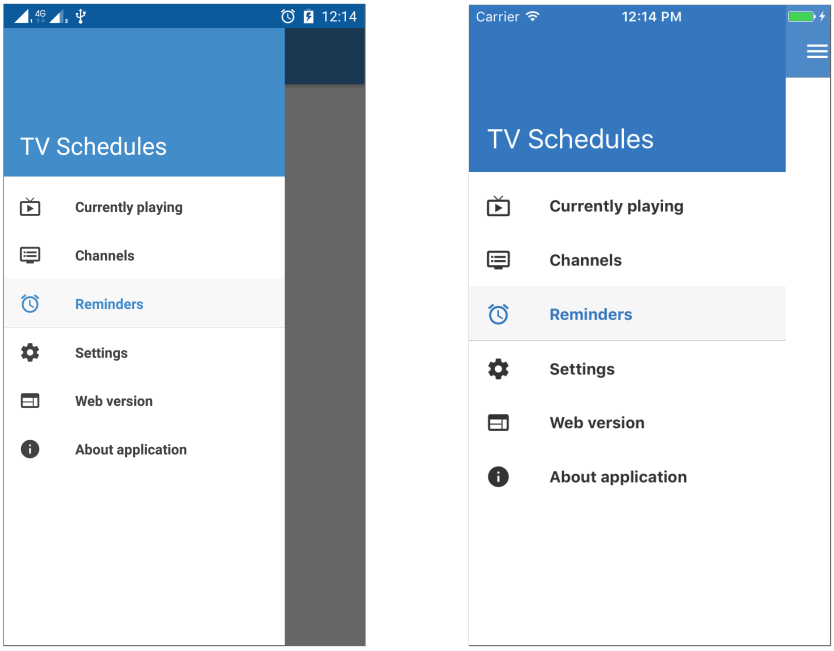


Android iOS

Slika 6.3: Uvodni zaslon z logotipom aplikacije, ki je prikazan zaradi inicializacije NativeScripta.

V okviru stranske vrstice se prikaže navigacijski meni s povezavami na strani za ogled trenutno predvajanih oddaj, seznama kanalov, seznama opo-

mnikov, nastavitev ter za odpiranje spletne različice aplikacije in informacij o sami aplikaciji (Slika 6.4). Po izbiri povezave se stranska vrstica zapre, prične pa se nalagati izbrana stran. Opozoriti je potrebno, da potek odpiranja strani deluje nekoliko drugače, kot smo vajeni iz razvoja domorodnih aplikacij. V prejšnjem podpoglavju 6.3 smo omenili, da se ponovno izriše celotna stran. To pomeni, da moramo namesto klica za odpiranje strani in zapiranja vrstice narediti obratno. Najprej zapremo vrstico, počakamo na konec animacije ter šele nato prožimo odpiranje nove strani. Stransko vrstico poleg naslovne lahko smatramo za najpomembnejši gradnik aplikacije, saj nam omogoča prehajanje med glavnimi stranmi.

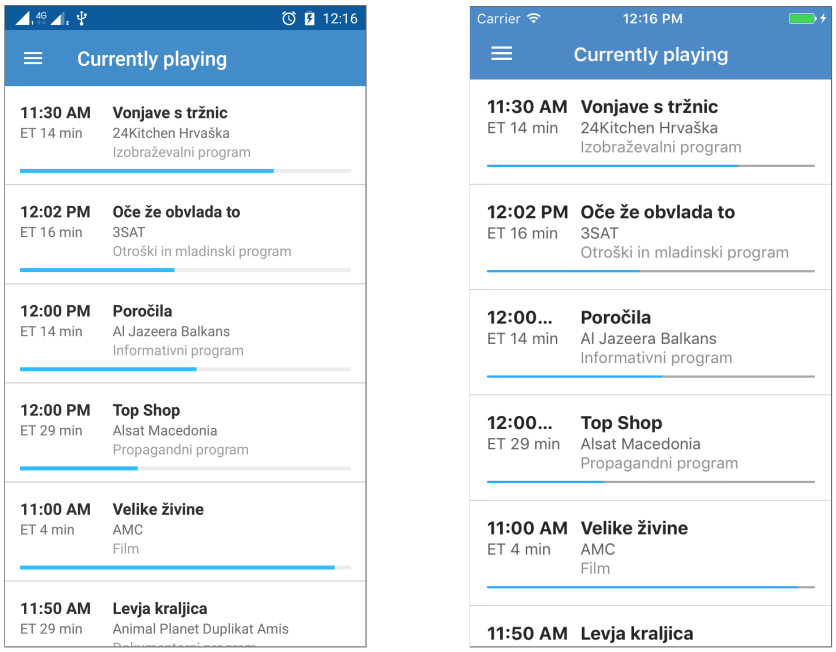


Android iOS

Slika 6.4: Stranska vrstica, ki vsebuje navigacijski meni s povezavami na strani aplikacije.

6.5.2 Spored

Prvo stran aplikacije predstavlja seznam oddaj, ki se trenutno predvajajo na kanalih dostopnih v aplikaciji (Slika 6.5). Za vsako oddajo so izpisani zgolj najpomembnejši podatki – ura predvajanja, naslov, kanal, kategorija in morebitni preostali čas. Za lažjo predstavo o trajanju oddaje je dodana vrstica napredka (ang. progress bar), ki označuje že pretekli del oddaje v primerjavi s preostalim.

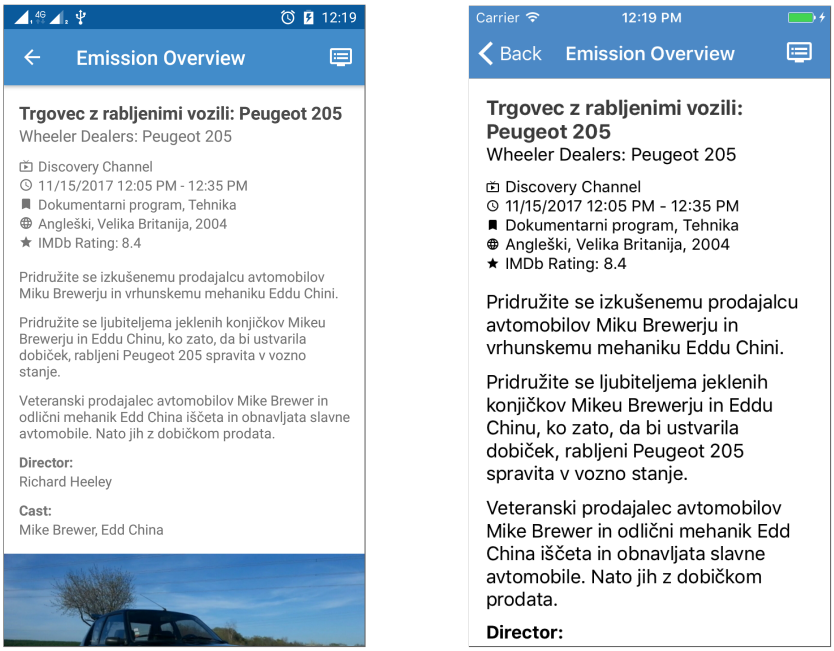


Android iOS

Slika 6.5: Stran s seznamom trenutno predvajanih oddaj na kanalih, ki se odpre kot prva stran aplikacije.

Ob pritisku na izbrano oddajo, se odpre nova stran s podrobnim pregledom njenih podatkov (Slika 6.6). Predstavlja jo komponenta *EmissionComponent*, ki je namenjena prikazu podatkov o oddaji glede na mesto odpiranja. V tem primeru smo jo odprli na strani trenutno predvajanih oddaj, zato se v njeni naslovni vrstici prikaže le možnost za prikaz sporeda na njenem ka-

nalu. V vsebinskem delu se izpišejo vsi znani podatki o oddaji – lokaliziran in izvorni naslov, kanal predvajanja (prikaže se samo ob odprtju iz omenjene strani), čas predvajanja, kategorija in žanr, jezik, država in leto produkcije, ocena IMDb, opis ter produkcijski podatki. V spodnjem delu se izriše tudi slika, ki ponazarja utrinek iz oddaje. Obe opisani strani uporabljata storitev *EmissionService*. Ob prikazu prve strani, se vanjo naložijo osnovni podatki oddaj. Po prehodu na drugo stran se naložijo podrobni podatki za izbrano oddajo. Vsi podatki so v storitvi shranjeni dokler ne zapustimo aplikacije.



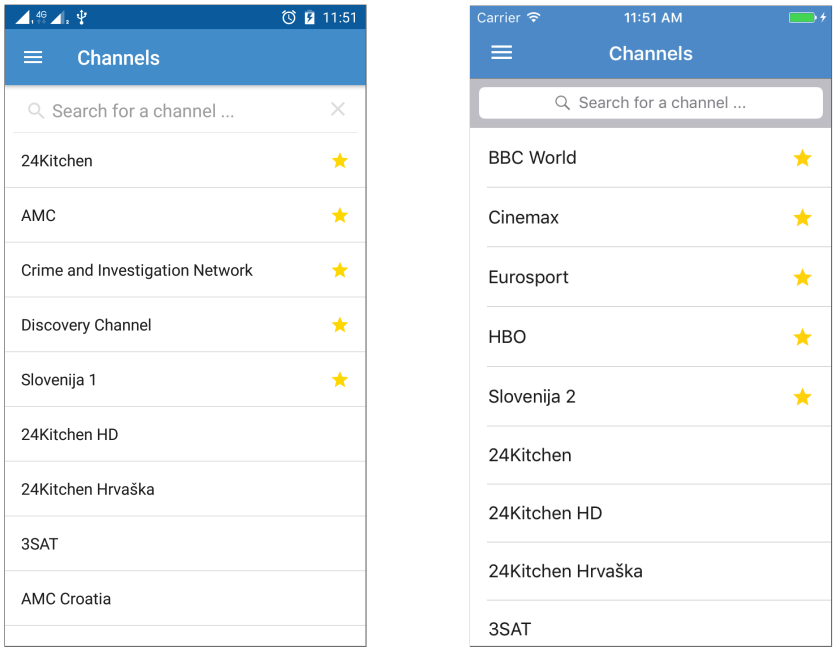
Android iOS

Slika 6.6: Stran s podrobnimi podatki oddaje, ki jo lahko odpremo iz različnih strani aplikacije.

6.5.3 Kanali

Pregled seznama vseh kanalov je možen na ločeni strani (Slika 6.7). Vseh skupaj je okrog 200 in so razvrščeni po abecedi. Shranjeni so v podatkovni bazi

aplikacije, ki se osvežuje enkrat dnevno. Za hitrejše iskanje je vključen tudi iskalnik. Priljubljeni kanali so označeni z zvezdico in se venomer nahajajo na vrhu seznama.

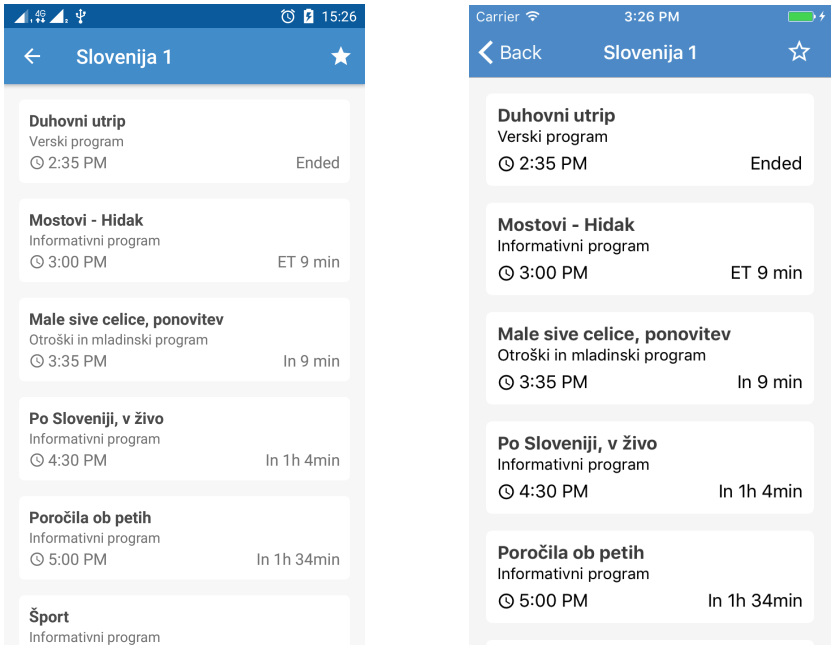


Android iOS

Slika 6.7: Stran s seznamom kanalov, po katerih je možno iskati in jih označiti kot priljubljene.

Ob izbiri določenega kanala se prikaže nova stran, ki prikazuje seznam oddaj na sporedu za tekoči programski dan (Slika 6.8). Izpišejo se zgolj osnovni podatki oddaj, tako kot pri strani s trenutno predvajanimi oddajami. V kolikor se je oddaja že zaključila, ima prikazan status o zaključku. V primeru, da se še ni začela, je prikazan čas do pričetka predvajanja. V naslovni vrstici je prikazana tudi ikona, s katero je možno označiti kanal kot priljubljen ali navaden. Po pritisku na izbrano oddajo, se odpre komponenta *EmissionComponent*, ki sproži nalaganje podrobnih podatkov oddaje. Tokrat je njen prikaz nekoliko spremenjen, saj se med podatki ne prikaže kanal predvajanja. Namesto ikone za pregled oddaj na kanalu, je prikazana možnost

za dodajanje oziroma brisanje opomnika – velja le za oddaje, ki se še niso pričele.



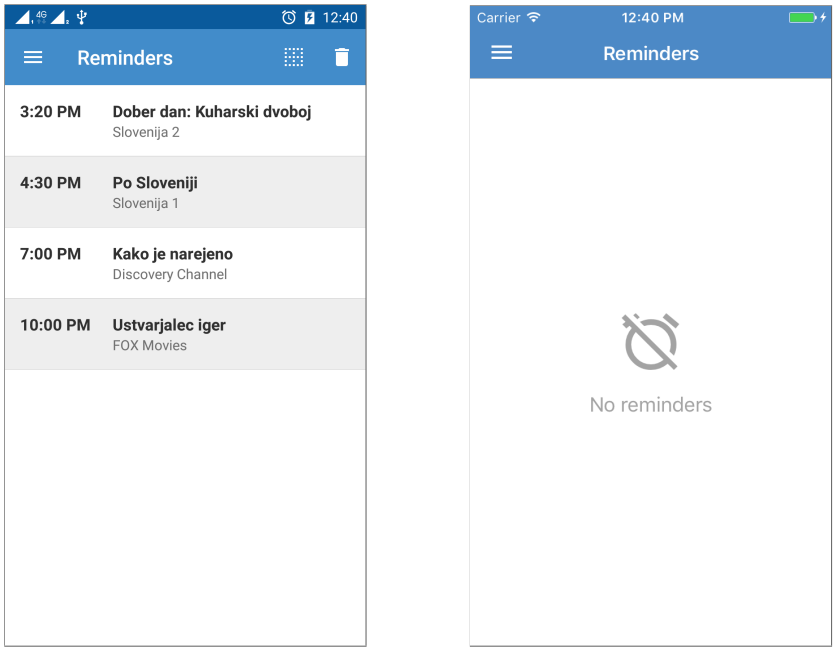
Android iOS

Slika 6.8: Stran s seznamom oddaj na sporedu kanala, ki imajo označeno stanje predvajanja.

6.5.4 Opomniki

Opomnik o predvajanju oddaje nastavimo v okviru komponente *Emission-Component*. Po pritisku ikone se prikaže obvestilo (ang. toast), ki sporoča uspešnost njegove nastavitve. V podatkovno bazo se shrani kanal, ura, naslov in oznaka oddaje, nakar pridobimo njegov identifikator. Nadzor nad opomniki aplikacije vrši knjižnica nativescript-local-notifications, ki izpostavlja metodi za dodajanje in brisanje. Pri dodajanju nastavimo identifikator, naslov, vsebino in čas prikaza opomnika. Za brisanje v metodi knjižnice in v bazi uporabimo le identifikator. Razlog za shranjevanje v bazo je v prikazu

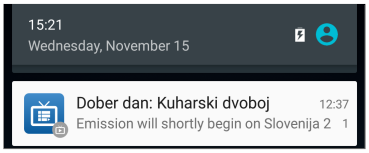
vseh opomnikov na svoji strani (Slika 6.9).



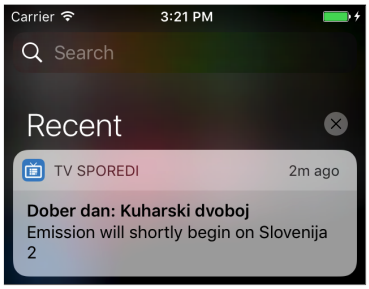
Android iOS

Slika 6.9: Stran s seznamom nastavljenih opomnikov, ki jih lahko izbrišemo.

Namesto komponente prikaznega seznama, ki jo zagotavlja jedro NativeScripta, smo uporabili napredni prikazni seznam iz knjižnice nativescript-telerik-ui. Razlikuje se po funkcionalnostih, saj poleg klasične izbire predmeta seznama podpira proženje operacij po pridržanju izbire (ang. long item press). To nam omogoča, da označimo več opomnikov hkrati in jih izbrišemo. Klasična izbira odpre stran s pregledom podatkov o oddaji, ki deluje na enak način, kot v okviru strani za prikaz oddaj na kanalu. V kolikor opomnika ne izbrišemo, se nekaj minut pred predvajanjem oddaje prikaže obvestilo v obvestilnem središču (ang. notification center) platforme, predvaja pa se tudi opozorilni zvok. Obvestilo je prikazano na sliki 6.10. Pretekli opomniki se iz baze pobrišejo ob naslednjem zagonu aplikacije, seznam vseh pa na podlagi poizvedbe SQL (ang. Structured Query Language) vedno prikaže zgolj prihodnje.



Android



iOS

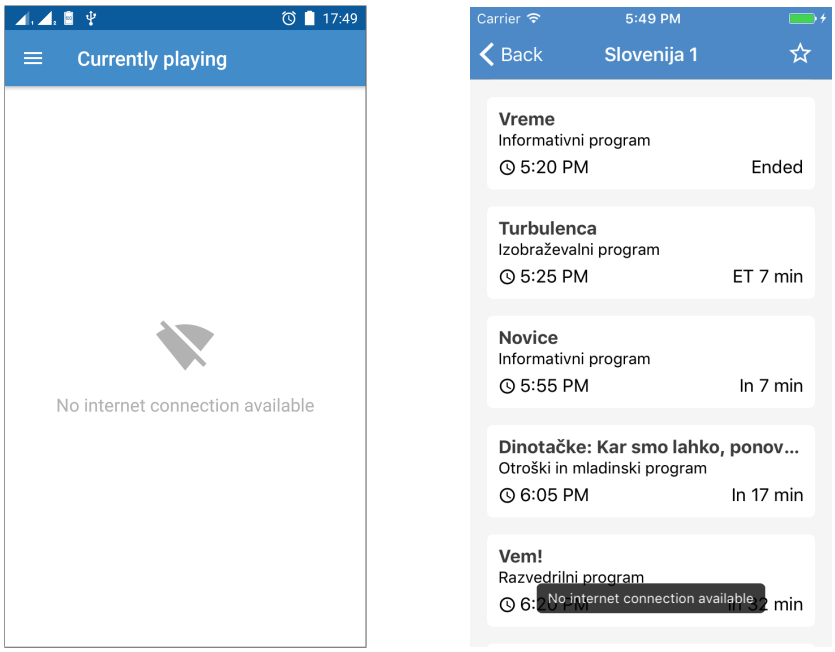
Slika 6.10: Obvestilo o pričetku predvajanja oddaje, ki se prikaže v okviru obvestilnega središča platforme.

6.6 Dodatni izzivi

Med razvojem smo naleteli na dodatne izzive, katerih rešitve predstavljajo vidne spremembe vmesnika aplikacije ali pa so uporabniku skrite in prinašajo splošne izboljšave. V nadaljevanju se bomo omejili na štiri najpomembnejše, ki vplivajo na uporabniško izkušnjo, hkrati pa so zanimivi iz razvojnega vidika.

6.6.1 Obnašanje v nepovezanem načinu

Občasno se zgodi, da mobilna naprava izgubi povezavo s podatkovnim omrežjem. V naši aplikaciji smo od tega močno odvisni, saj se z izjemo kanalov in opomnikov podatki ne shranjujejo v pomnilnik naprave. Razlog za opuščanje shranjevanja vseh je njihova količina. Pri večjem številu kanalov bi zapisovanje vsakega izmed sporedov vzelo kar nekaj časa, kasneje pa bi bilo treba izbrisati še pretekle podatke. Posledično je enostavneje, da jih vsakokrat prenesemo znova, obenem pa nam ni potrebno skrbeti za ažurnost na napravi. Rešitev za čas, ko ni mogoče pridobiti podatkov je poseben zaslon, ki prikazuje odsotnost podatkovnega omrežja. V primeru, ko uporabnik že vidi podatke in izgubi povezavo, se mu prikaže obvestilo o nedostopnosti do interneta. Obe možnosti sta zajeti na sliki 6.11.



Android iOS

Slika 6.11: Na Androidu je prikazan poseben zaslon, ki označuje odsotnost internetne povezave. Na iOSu je prikazano obvestilo o izgubi povezave.

Zaznavanje nepovezanosti je delo NativeScripta, ki zagotavlja modul `connectivity`. Z njim ugotovimo tip povezave – brezžična, mobilna, brez. Dodatno izpostavlja metodo `startMonitoring`, kateri podamo povratni klic, ki se izvede ob spremembi tipa povezave. Za poenostavitev spremljanja sprememb smo izdelali dodatno storitev z nastavitvami *SettingsService*, v katero smo dodali predmet obnašanja (ang. behavior subject) `isInternetAvailableSource`. Poleg njega smo vključili dodatne metode za pridobivanje, ki vračajo opazovance z ustreznim stanjem – ali je internet dosegljiv ali ni. Nanje se naročimo glede na potrebe v drugih opazovancih. Napisali smo še posebno metodo `checkInternetAvailable`, ki ugotovi ali tip povezave ustreza dosegljivosti interneta. Primer zaznavanja načina povezanosti je prikazan v izseku programske kode 6.2, ki se sicer nahaja v konstruktorju storitve z nastavitvami.

```
connectivity.startMonitoring(newConnectionType =>
  this.isInternetAvailableSource.next(
    this.checkInternetAvailable(newConnectionType)
  )
);
```

Izsek programske kode 6.2: Nastavljanje stanja povezave s pomočjo predmeta in metode povezavnega modula, ki sporoča spremembe.

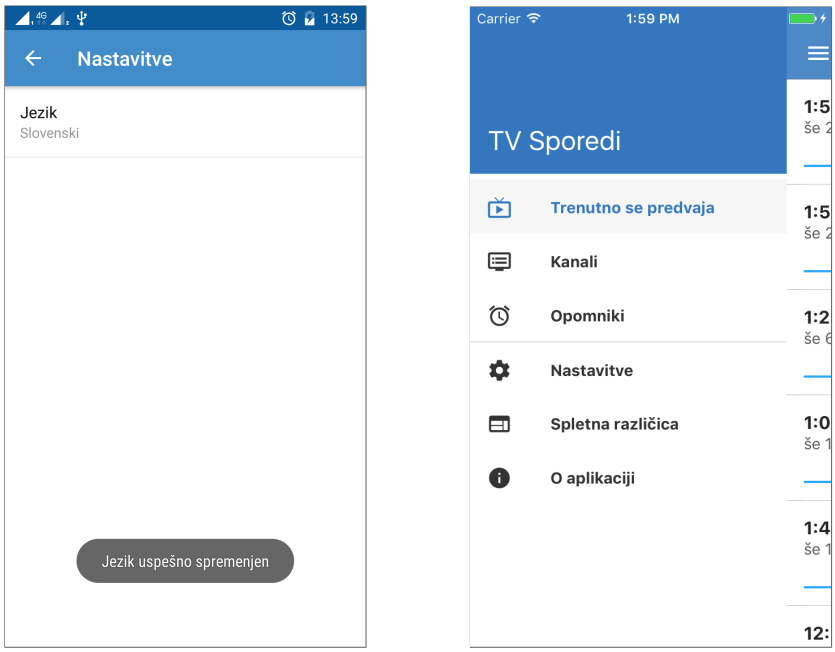
S takšnim pristopom smo ločili logiko zaznavanja v svojo storitev in poenostavili njeno morebitno spreminjanje.

6.6.2 Večjezičnost vmesnika

Podporo večjezičnosti vmesnika smo dodali že v samem začetku razvoja, saj se podatki sporeda vnašajo v več jezikih. Posledično je smiselno, da se jezik vmesnika prilagodi uporabniku in kjer je mogoče tudi podatki. Trenutni API tega sicer ne podpira, zato so vsi podatki v slovenščini. V kolikor se bo v prihodnosti podprlo večjezičnost podatkov, bo sprememba v okviru aplikacije trivialna. V komunikacijsko storitev, kjer se nahaja glava zahtevka HTTP (ang. HTTP request headers), bi bilo potrebno dodati le polje z jezikom.

Jezik aplikacije se poskusi nastaviti na jezik operacijskega sistema takoj ob njenem zagonu. V primeru, da ni vsebovan med datotekami s prevodi, se uporabi privzeti jezik, ki je nastavljen na angleščino. Navkljub temu je na strani z nastavitvami še vedno mogoče prilagoditi jezik. Spremeni se takoj po izbiri in se shrani v nastavitve aplikacije. Uporabljen je tudi ob naslednjem zagonu. Slika 6.12 prikazuje stran z nastavitvami in primer prikaza slovenskega vmesnika.

Izbira načina vstavljanja prevodov je bila dokaj enostavna. NativeScript zaenkrat še ne podpira prevodov v predlogah s pomočjo značke `i18n`, ki izhaja iz Angularja. Razmišljali smo o izdelavi lastne storitve, ki bi skrbela za zagotavljanje nizov, na voljo pa smo imeli tudi knjižnico ng2-translate, narejeno za Angular. Odločili smo se za slednjo, saj ne predstavlja težav pri prevajanju v načinu AOT. Deluje na osnovi datoteke JSON, v katero



Android iOS

Slika 6.12: Uporabniški vmesnik v slovenščini. Na Androidu je prikazana stran z nastavitvami in obvestilo o ravnokar spremenjenem jeziku. Na iOSu je prikazana stranska vrstica ob kateri je odprta stran s seznamom trenutno predvajanih oddaj.

zapišemo ključe in vrednosti (prevode). Datoteko umestimo v mapo `i18n`, poimenovana pa mora biti z dvočrkovno kodo jezika. Za prikaz nizov v predlogah že vključuje cev `translate`, ki jo uporabimo v interpolaciji. Primer uporabe je prikazan v izseku programske kode 6.3. Prednost knjižnice je pridobivanje prevodov s pomočjo opazovancev. To pomeni, da se ob zamenjavi jezika takoj sproži posodobitev vseh nizov, česar rezultat je v trenutku spremenjen jezik aplikacije.

```
<SearchBar [hint]="'CHANNELS.SEARCH_TEXT' | translate">
</SearchBar>
```

Izsek programske kode 6.3: Primer uporabe cevi `translate`, ki zagotovi prevod za izbran jezik na podlagi ključa. Med drugim je uporabljena za prikaz privzetega niza v iskalni vrstici na strani s seznamom kanalov.

6.6.3 Vključevanje podatkovne baze

Podpora funkcionalnosti opomnikov je zahtevala vključitev podatkovne baze v mobilno aplikacijo. Pred dejansko vključitvijo smo razmišljali o alternativni rešitvi, ki bi nadomestila relacijsko bazo in podatke hranila na delno strukturiran način. Odločili smo se, da zapisovanje podatkov prepustimo obstoječi knjižnici nativescript-sqlite ter se raje osredotočimo na boljšo podporo načinu pomnenja podatkov v aplikaciji. Obenem smo želeli čimbolje izkoristiti bazo ter vanjo shraniti vse možne podatke, ki so smiselni za daljše hranjenje. Posledično smo poleg opomnikov v bazo vključili še kanale. Dodajanje podatkovne baze v aplikacijo smo izvedli v dveh korakih.

Prvi korak je obsegal namestitev knjižnice, dodajanje definicije tabel podatkovne baze in izdelavo glavne storitve podatkovne baze *DatabaseService*. Knjižnica ovija dostop do domorodnih implementacij baze SQLite in izpostavlja nekaj metod s katerimi vršimo operacije nad njo. Ena izmed teh je `execSQL`, ki omogoča izvajanje poizvedb SQL. Za njeno izvršitev moramo imeti vzpostavljeno povezavo z bazo. Celotno logiko delovanja smo implementirali v storitev *DatabaseService*, ki hrani primerek (ang. instance) podatkovne baze s povezavo, zagotovljen s strani knjižnice. Hkrati omogoča osnovne operacije nad določeno podatkovno bazo, kot so vstavljanje, posodabljanje, brisanje in branje podatkov iz njenih tabel. Vse uporabljajo prej omenjeno metodo, v katero podajo ustrezno razčlenjeno poizvedbo. Glede na to, da se implementaciji baze SQLite na obeh platformah razlikujeta, moramo paziti, da so vse operacije in vrednosti kompatibilne med njima. Tako smo ločili operacije specifične za to vrsto podatkovne baze v svojo storitev.

Zaradi tega v preostalem delu aplikacije ne pišemo poizvedb, temveč le zagotovimo, da metode storitve *DatabaseService* kličemo s pravilno podanimi parametri.

Drugi korak je zajel implementacijo operacij nad podatkovnimi objekti in konfiguracijo baze. Razmišljali smo o izdelavi lastnega ORM (ang. Object-Relational Mapping), a za dve tabeli predstavlja zgolj dodaten nivo kompleksnosti. Prav tako bi bilo potrebno zajeti vse možne scenarije operacij že v storitvi podatkovne baze. Namesto tega smo se odločili za način preslikave podatkov, nad katerim bomo imeli poln nadzor. Izdelali smo generičen razred *Repository*, ki služi kot osnova za implementacijo vzorca repozitorija (ang. repository pattern). Z njim smo določili katere metode mora vsebovati končni repozitorij. Prikazane so v izseku programske kode 6.4. Vsak končni repozitorij, ki razširja generičnega, določa preslikavo med objektom in polji tabele. Objekt je določen s podatkovnim modelom, zato lahko enostavno dostopamo do njegovih lastnosti. Repozitorij za izvedbo operacij na lastni tabeli uporabi metode storitve *DatabaseService*.

```
export abstract class Repository<T> {

  constructor(
    protected dbService: DatabaseService
  ) {}

  abstract save(entity: T);
  abstract saveAll(entityArray: Array<T>);
  abstract remove(entity: T);
  abstract find(entity: T);
  abstract findAll();

}
```

Izsek programske kode 6.4: Generičen razred repozitorija, ki določa metode za implementacijo v končnem repozitoriju.

Pred dejansko uporabo podatkovne baze, se mora ta najprej inicializirati. Storitev *DatabaseService* ob prvem klicu operacije pridobi primerek s

povezavo. Takoj zatem se izvedejo poizvedbe, ki na primer zagotovijo obstoj tabel ali izvedejo operacije nad zapisi. Del inicializacije je konfiguracija, ki določa imena tabel in pripadajoča polja. Vključuje tudi poizvedbe, ki se morajo izvesti ob prvem dostopu. Nanjo se sklicujejo vsi končni repozitoriji, saj dostopajo do imen tabel in polj, ki so vsebovana v ločenih objektih. Z njimi zagotovimo, da so vsa imena na enem mestu in nadzor nad spremembami ključev objektov. V primeru, da v končnem repozitoriju ob znatnih spremembah objekta pozabimo spremeniti ključe, prevajanje aplikacije ne bo uspešno. S takšnim pristopom ločimo definicijo tabele od modelov, kar v primeru uporabe ORM ne bi bilo povsem mogoče. Tako je delo s podatkovno bazo omejeno na repozitorije in storitev baze, preostali del aplikacije pa se njenega obstoja sploh ne zaveda. Primer objektov v konfiguracijski datoteki je prikazan v izseku programske kode 6.5.

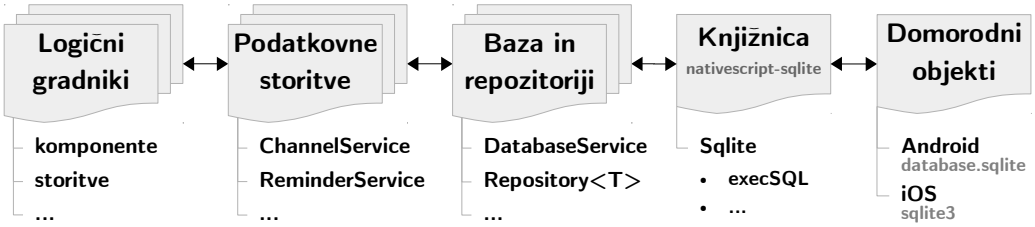
```
export const REPOSITORY_CHANNEL = {
  TABLE: "channels",
  COLUMNS: {
    ID: "id",
    SID: "sid",
    NAME: "name",
    FAVORITE: "favorite"
  }
};

export const REPOSITORY_INIT = {
  CHANNEL:
  `CREATE TABLE IF NOT EXISTS ${REPOSITORY_CHANNEL.TABLE} (
  ${REPOSITORY_CHANNEL.COLUMNS.ID} INTEGER UNIQUE NOT NULL,
  ${REPOSITORY_CHANNEL.COLUMNS.SID} VARCHAR(255) UNIQUE NOT
     NULL,
  ${REPOSITORY_CHANNEL.COLUMNS.NAME} VARCHAR(255) NOT NULL,
  ${REPOSITORY_CHANNEL.COLUMNS.FAVORITE} INTEGER(1) DEFAULT 0
  );`,
  ...
```

```
};
```

Izsek programske kode 6.5: Konfiguracija repozitorijev, ki vključuje izvleček definicije tabel in inicializacijskih poizvedb.

Za podporo operacijam nad podatkovno bazo za kanale in opomnike, smo izdelali repozitorija *ChannelRepository* in *ReminderRepository*. Oba sta vključena samo v pripadajoči podatkovni storitvi *ChannelService* in *ReminderService*. Gradniki, ki vsebujejo logiko za prikaz in upravljanje podatkov, uporabljajo le podatkovne storitve, saj so te zadolžene za združevanje podatkov iz APIja in baze. Repozitoriji tako niso izpostavljeni logičnim gradnikom, temveč je med njimi dodatna plast, ki skrbi za vse podatkovne operacije. Slika 6.13 prikazuje opisane nivoje.



Slika 6.13: Arhitektura pretoka podatkov v aplikaciji z vključeno podatkovno bazo.

Poskus optimizacije operacij podatkovne baze

Pri prvi uporabi aplikacije in dodajanju vseh kanalov v bazo, smo naleteli na dodatno težavo. Vnesti je potrebno okrog 200 kanalov, kar je lahko precej zamudno delo. Naš prvi pristop je temeljil na posamičnem dodajanju, torej en klic operacije dodajanja na kanal. Izvajanje operacij je potekalo zelo počasi, kar se je poznalo na odzivnosti uporabniškega vmesnika. Težavo smo zaznali samo na operacijskem sistemu Android, medtem ko je aplikacija na iOSu delovala tekoče. Razmišljali smo v smeri odcepitve operacij podatkovne baze od glavne niti JavaScripta v ločeno. To bi zagotovo pohitrilo njihovo

delovanje, hkrati pa omogočilo nemoteno uporabo vmesnika. NativeScript podpira API za izvajanje težavnejših operacij v ločeni niti, bolje poznan pod imenom *Web Workers*[1]. Odločili smo se, da storitev *DatabaseService* spremenimo v takšno obliko, da bo za izvajanje vseh operacij uporabila ločeno nit. Na vrhnji nivo imeniške strukture projekta smo dodali dve enoti. Prva, imenovana `helpers`, v kateri se nahajajo pomagala, je namenjena funkcijam, ki pripomorejo k izvajanju neke operacije. Vanjo smo umestili funkcijo za pridobivanje primerka baze s povezavo. Druga, imenovana `workers`, v kateri se nahajajo naloge za izvajanje v ločeni niti, služi komunikaciji z glavno nitjo in izvajanju nalog. Vsebuje nalogo, ki pridobi primerek baze in izvede poizvedbo z metodo `execSQL`. Celotno delovanje je osnovano na obljubah (ang. promise), ki so del JavaScripta. Za pričetek izvajanja v ločeni niti, najprej inicializiramo objekt `Worker`, kateremu podamo pot do naloge. Nato nastavimo funkciji za povratna klica `onmessage` in `onerror`, ki sta namenjeni pridobitvi rezultata oziroma napake. Za konec sprožimo izvajanje naloge z metodo `postMessage`, v katero podamo podatke. Ti se v pravilno obliko, ki jo razume tudi vhodni del naloge, pretvorijo s pomočjo metode `prepareMessage`. V okviru poizvedbe SQL smo podali tabelo vsebovano v spremenljivki `table`. Imena njenih polj, v katere napolnimo podatke, smo podali v spremenljivki `parsedParams` in pripadajoče vrednosti v spremenljivki `parsedValues`. Primer klica za vstavljanje podatkov v bazo je prikazan v izseku programske kode 6.6. Podatki, ki jih pošiljamo v ločeno nit tja dospejo v obliki JSON. To pomeni, da lastno snovanih objektov ne moremo poslati.

```
return new Promise((resolve, reject) => {
  let worker = new Worker("../../workers/database/execute");
  worker.onmessage = (res) => resolve(res.data);
  worker.onerror = (err) => reject(err);
  worker.postMessage(this.prepareMessage([
    `INSERT INTO ${table} ${parsedParams}
     VALUES ${parsedValues}`,
```

[1]NativeScript Documentation – Multithreading Model. URL: `https://docs.nativescript.org/core-concepts/multithreading-model`. [Dostopano: 18.11.2017]

```
    values
  ]));
});
```

Izsek programske kode 6.6: Obljuba, ki vsebuje operacijo za dodajanje podatkov v bazo in se izvede v ločeni niti. Vrnemo jo kot rezultat operacije, da lahko na osnovi njenega rezultata izvedemo nadaljnja dejanja.

Po končani implementaciji smo preizkusili hitrost delovanja operacij. Rezultati so bili slabi, saj se je čas izvajanja podaljšal celo pri branju podatkov. Ugotovili smo, da izvajanje v ločeni niti pred izvršitvijo naloge opravi precej režijskega dela in kot tako ni namenjeno krajšim operacijam oziroma zaporedju njih. Spremembe niso bile opazne niti v odzivnosti vmesnika. Zaradi tega smo dodali dodatno operacijo dodajanja, ki je namenjena skupinski (ang. batch) obdelavi. Izkoristili smo stavek za dodajanje v poizvedbi SQL, ki omogoča vnos več zapisov hkrati. Ob prvem dodajanju kanalov nas je presenetila dodatna težava. Metoda knjižnice `execSQL` vrednosti pričakuje kot ločen argument, dejansko pojavitev v poizvedbi pa označimo z znakom `?`. Omejitev števila vrednosti za posamezno poizvedbo je 999, kar je naša poizvedba krepko presegala. Število omejitve je določeno v domorodni implementaciji baze na obeh platformah. Razlog za omejevanje je poraba delovnega pomnilnika, kamor se za čas izvedbe shranijo vrednosti. Posledično smo dodajanje izvedli v korakih. Tako se izvede toliko poizvedb z največ 999 vrednostmi, kolikor jih je potrebno za dodajanje vseh zapisov. Pri tem je potrebno upoštevati, da ima en zapis v bazi več vrednosti, ki pritičejo določenemu polju. Na osnovi tega smo izračunali, koliko zapisov na poizvedbo oziroma skupino lahko dodamo. Izračun je enostaven – `x` skupin dobimo na osnovi količnika med največ vrednostmi in števila polj `y`, ki pripadajo enemu zapisu ($x = \frac{999}{y}$). Sledi le še izvajanje poizvedb, ki ga vnovič opravimo z obljubami. Primer razdelitve in načina izvajanja poizvedb je prikazan v izseku programske kode 6.7. V njej spremenljivka `batchSize` predstavlja število skupin, `values` je polje z vrednostmi, `db` predstavlja primerek baze, funkcija `insertBatch` pa poskrbi za pravilno oblikovanje poizvedbe.

Funkciji `resolve` in `reject` zagotovi obljuba, v kateri se nahaja omenjena koda.

```
let batches: Array<Promise<any>> = [];
for (let i = 0; i < values.length; i += batchSize) {
  batches.push(
    insertBatch(db, values.slice(i, i + batchSize))
  );
}
Promise.all(batches).then(
  res => resolve(res),
  err => reject(err)
);
```

Izsek programske kode 6.7: Razdelitev vrednosti v skupine in klic operacije za izvajanje poizvedb s pomočjo obljub.

Izvajanje dodane operacije v ločeni niti ni doprineslo k izboljšavi hitrosti. Idejo odcepitve operacij od glavne niti smo zato opustili, vendar kljub temu uporabili novo operacijo. Bistvenih sprememb med posamičnim in skupinskim dodajanjem v Androidu ni opaziti, medtem ko iOS kanček hitreje prikaže celoten seznam kanalov po dodajanju. Sklenili smo, da je težava bodisi v načinu delovanja NativeScripta na Androidu bodisi v uporabljeni knjižnici.

6.6.4 Prevajanje in učinkovitost aplikacije

Aplikacija temelji na Angularju, zato se način prevajanja ujema s spletno aplikacijo. Za čas razvoja smo uporabljali JIT, za pripravo distribuirane različice pa AOT. NativeScript privzeto ne podpira prevajanja v načinu AOT, zaradi česar je potrebno namestiti vtičnik nativescript-dev-webpack. Ta zahteva dodajanje glavne vhodne datoteke, preko katere se aplikacija po prevajanju z AOT zažene. Vtičnik prinaša dodatna orodja za optimizacijo kode in pohitritev izvajanja aplikacije. Kodo spremeni v gršo obliko (ang. uglify), kar pomeni da iz nje odstrani vse nepotrebne znake, skrajša imena spremenljivk in podobno. S tem zmanjša velikost kodne osnove aplikacije. Pohi-

tritev izvajanja aplikacije je zaenkrat omejena samo na Android. Bistveni doprinos k temu naredi posnetek dela pomnilnika v kontekstu navideznega stroja V8 (ang. V8 heap snapshot), ki se shrani v distribuirano različico[2]. Ob odprtju aplikacije se tako najprej naloži posnetek, kar skrajša čas nalaganja aplikacije. Poleg optimizacij vtičnik ponuja možnost priprave različice za objavo, ki jo je potrebno podpisati z ustreznim ključem. Po uspešnem postopku ustvari datoteko, ki se jo lahko naloži v trgovino aplikacij za posamezno platformo.

Učinkovitost glede na način prevajanja

Priprava distribuirane različice ne zahteva uporabe načina prevajanja AOT. Uporabimo lahko JIT in poleg kodne osnove v datoteko zapakiramo še prevajalnik. Zaradi zanimanja o dejanski razliki med obema načinoma, smo izvedli primerjavo na obeh platformah. Merili smo čas nalaganja strani aplikacije in preverili velikost zasedenega prostora po namestitvi. Uporabili smo mobilni telefon Lenovo Vibe Shot Z90 z nameščenim Androidom 6.0.1 in v simulatorju delujoč iPhone SE z iOS 10.3. Primerjava je bila izvedena na osnovi realnega scenarija (ang. real world scenario benchmark), kar pomeni da smo čas merili ročno, brez pomoči orodij za avtomatizacijo. Te nad aplikacijo dodajo plast, ki nadomešča uporabniške operacije in bi lahko dodatno vplivala na hitrost izvajanja. Merjenje časa smo izvedli z vzporednim zagonom štoparice ob pritisku izbrane možnosti v uporabniškem vmesniku. Med obema načinoma je bilo izvedeno v enakih pogojih. Pri tem je bila edina odprta aplikacija zgolj ta, ki smo jo preizkušali (poleg sistemskih, ki se morajo izvajati). Pri prevajanju z AOT smo vključili tudi dodatne (v začetku podpoglavja opisane) zmožnosti, ki zmanjšajo velikost aplikacije in pohitrijo njeno izvajanje. Za primerjavo smo izbrali tri strani. Uvodni zaslon skupaj s seznamom trenutno predvajanih oddaj, kjer smo počakali do prikaza podat-

[2]NativeScript Documentation – Using Webpack to Bundle Your Code. URL: `https://docs.nativescript.org/best-practices/bundling-with-webpack`. [Dostopano: 18.11.2017]

kov. Stran s kanali, kjer smo prav tako počakali do prikaza vseh. Za konec smo izmerili še čas nalaganja strani s podatki o oddaji, kjer smo počakali do prikaza njene slike. Čas nalaganja posamezne strani smo izmerili v treh poskusih, pred katerimi smo vsakič pobrisali podatke in začasne datoteke aplikacije. Vse tri smo na koncu uporabili za izračun povprečja. Rezultati izmerjenega časa v sekundah so prikazani v tabeli 6.1.

	Android		**iOS**	
	JIT	AOT	JIT	AOT
Uvodni zaslon	13,17	7,14	5,66	4,17
Seznam kanalov	3,48	2,77	1,29	1,08
Podatki o oddaji	3,21	2,35	2,01	1,84

Tabela 6.1: Rezultati učinkovitosti aplikacije za Android in iOS, ki je bila prevedena v načinih JIT in AOT. Časi so podani v sekundah. Nakazujejo koliko uporabnik v povprečju čaka za prikaz podatkov na posameznih straneh v vsakem izmed načinov.

Izmerjeni časi so na prvi pogled dolgi, vendar je odzivnost aplikacije z izjemo uvodnega zaslona povsem primerljiva z drugimi. Na rezultate vpliva tudi hitrost nalaganja podatkov s strani REST APIja ter hitrost medmrežne povezave. Pri načinu prevajanja je potrebno upoštevati tudi to, da JIT aplikacija ne deluje povsem tekoče. Najverjetneje na odzivnost vpliva prav prevajanje v času izvajanja. V nasprotju s tem AOT deluje sprejemljivo. Omeniti je potrebno tudi razliko med Androidom in iOS v času prikaza uvodnega zaslona. Pri prvem je ta prikazan večidel časa, šele kasneje se prikaže vrstica napredka, ki nakazuje prenos podatkov iz zaledja za prvo stran. Pri drugem se v načinu JIT uvodni zaslon skrije nekje na polovici izmerjenega časa, v AOT pa je prikazan zgolj nekaj trenutkov. Način prevajanja med drugim vpliva na velikost nameščene aplikacije in podatkov, ki jih ta potrebuje za delovanje. Velikosti obeh načinov v megabajtih so prikazane v tabeli 6.2.

	Android			iOS		
	Aplikacija	Podatki	Skupaj	Aplikacija	Podatki	Skupaj
JIT	49,3	52,4	101,7	105,0	0,05	105,05
AOT	30,1	11,3	41,4	59,0	0,05	59,05

Tabela 6.2: Prostor, ki ga aplikacija zavzema po namestitvi glede na način prevajanja. Velikosti so podane v megabajtih. Aplikacija predstavlja izvršljivi del, ki ga sistem potrebuje za izvajanje. Podatki so namenjeni samemu delovanju aplikacije in ne vključujejo predpomnjenih datotek.

Na Androidu med podatki najdemo datoteke, ki omogočajo pohitritev delovanja aplikacije. Za razliko iOS hrani le podatkovno bazo, ki po vnosu vseh kanalov zaseda 50 KB. Poleg aplikacije in podatkov je potrebno upoštevati, da se kot predpomnjene datoteke shranijo še slike in odzivi zaledja v obliki JSON. Njihova velikost je v primerjavi s skupno zelo majhna, vendar ogromno pripomorejo k hitrejšemu nalaganju strani v aplikaciji.

Primerjava med načini prevajanja JIT in AOT potrjuje, da je med razvojem nekaj časa smiselno vložiti v podporo AOT. Rezultati so vidni tako v okviru uporabniške izkušnje, kot tudi na prostoru, ki ga celotna aplikacija zavzema.

Učinkovitost v primerjavi z domorodno aplikacijo

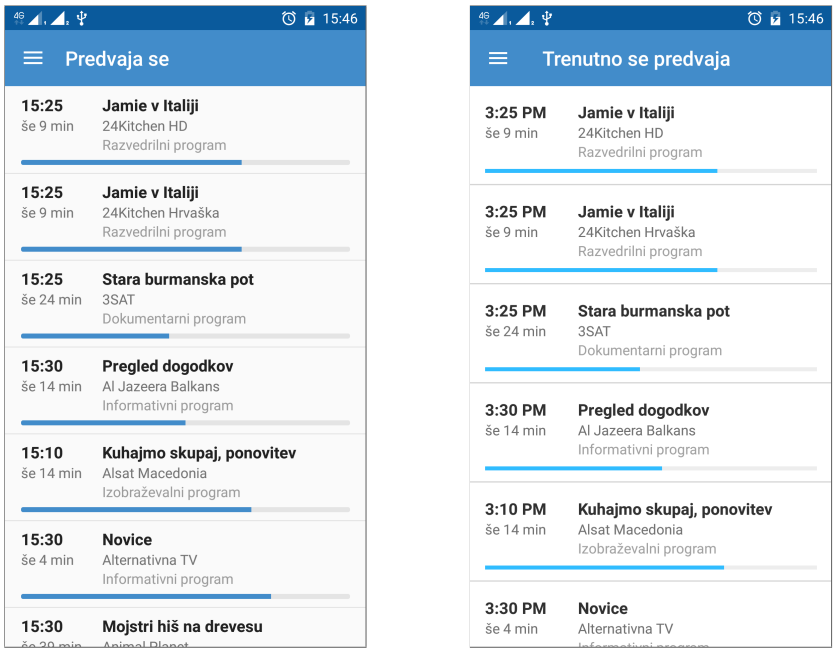
Način prevajanja v veliki meri vpliva na učinkovitost aplikacije. AOT poskrbi, da je uporabniška izkušnja na pričakovanem nivoju. Zanimalo nas je, kako se aplikacija izdelana na osnovi NativeScripta (AOT) obnese v primerjavi z domorodno. V ta namen smo izvedli preizkus na operacijskem sistemu Android, kjer smo poleg že opisane aplikacije uporabili prototip izdelan z domorodnimi tehnologijami. iOS smo tokrat izpustili, saj smo ocenili, da je delovanje aplikacije že na nivoju domorodne. Cilj primerjave je bil ugotoviti

čas nalaganja strani aplikacije, velikost namestitvenega arhiva aplikacije in velikost zasedenega prostora po namestitvi.

Prototipna aplikacija je bila izdelana v programskem jeziku Java, ki je uporabljen za podporo logičnemu delu. Za postavitev gradnikov na straneh je bil uporabljen označevalni jezik XML. Aplikacija izrablja funkcionalnosti, ki so dosegljive v Android API 23, podporo starejšim različicam pa zagotavljajo kompatibilnostne knjižnice. Za komunikacijo preko HTTP uporablja knjižnico Android Volley. Prototip zajema sklopa sporedov in kanalov. Implementirano ima tudi podatkovno bazo SQLite, v katero se shranjujejo kanali. Vmesnik je zelo podoben aplikaciji izdelani z NativeScriptom, le da ponekod uporablja naprednejše gradnike. Na sliki 6.14 je prikazana stran s trenutno predvajanimi oddajami v okviru obeh aplikacij, ki sta povsem primerljivi. Največja razlika med njima je v prikazu strani s podrobnostmi o oddaji. Domorodna aplikacija v naslovni vrstici prikaže sliko, ki tam ostane vse dokler se ne pomaknemo do konca vsebine. V aplikaciji izdelani z NativeScriptom to privzeto ni mogoče, zato je slika prikazana na dnu strani. Izgled te strani v obeh aplikacijah je prikazan na sliki 6.15. V domorodno aplikacijo ni vključen uvodni zaslon, saj se ga iz vidika uporabniške izkušnje ne potrebuje. Prav tako nima posebne inicializacije, zaradi katere bi bil sicer uporabljen.

Primerjavo smo izvedli na dveh mobilnih napravah. Prva, Lenovo Vibe Shot Z90, spada v višji cenovni razred in zastopa nivo zmogljivejših telefonov. Nameščen ima Android 6.0.1. Druga, Lenovo A1000, spada v najnižji cenovni razred in predstavlja nivo manj zmogljivih mobilnih naprav. Nameščen ima Android 5.0. Napravi različnih cenovnih razredov smo izbrali tudi z namenom, da vidimo kakšna je razlika med njima v hitrosti aplikacije izdelane z NativeScriptom. Pogoji za merjenje so bili pri obeh napravah enaki – obe sta bili povezani na isto omrežje in imeli odprto zgolj aplikacijo za katero smo merili čas (poleg sistemskih, ki morajo delovati).

Ciljne strani primerjave so ostale iste kot pri primerjavi učinkovitosti glede na način prevajanja, ki smo jo izvedli v prejšnjem podpoglavju 6.6.4.

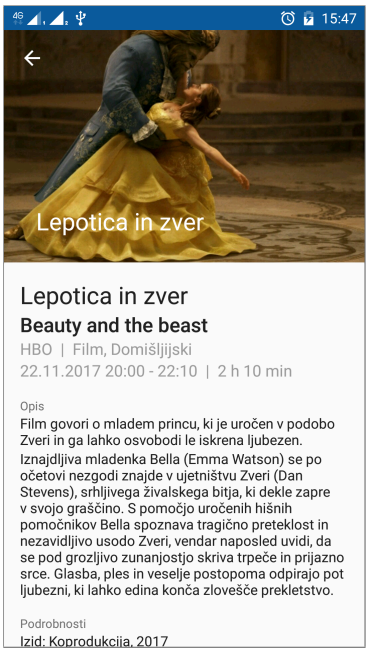


Domorodna NativeScript

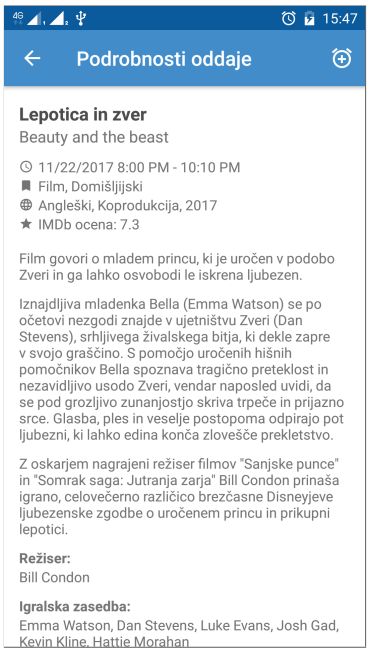
Slika 6.14: Stran s seznamom trenutno predvajanih oddaj, prikazana v okviru domorodne aplikacije in aplikacije osnovane na NativeScriptu.

Primerjavo smo zopet izvedli na podlagi realnega scenarija. Merjenja smo se zaradi večjih časovnih razlik na nekaterih straneh, lotili v dveh delih. V prvem smo merili hitrost delovanja aplikacije ob prvem zagonu oziroma po izbrisu podatkov aplikacije in predpomnjenih datotek. Izmerjeni časi (v sekundah) v tabeli 6.3 temeljijo na povprečju dveh merjenj, ki so povečini odstopala zgolj za nekaj desetink sekunde. V drugem delu smo izvedli še merjenje hitrosti, ko je bila aplikacija pred tem vsaj enkrat odprta – torej je podatke že imela shranjene. Rezultati v sekundah so prikazani v tabeli 6.4.

Bistvena razlika med aplikacijama se kaže ravno pri uvodnem zaslonu. Pri aplikaciji osnovani na NativeScriptu je bilo na napravi Z90 potrebno počakati



Domorodna



NativeScript

Slika 6.15: Stran s podrobnostmi o oddaji, prikazana v okviru domorodne aplikacije in aplikacije osnovane na NativeScriptu.

skoraj še enkrat toliko, kot pri domorodni. Na A1000 je ta čas približno potrojen. Stran s seznamom kanalov se ob prvem zagonu počasneje naloži v domorodni aplikaciji, česar vzrok je dodajanje kanalov v podatkovno bazo. Ob ponovnem odpiranju strani pa je počasnejša aplikacija osnovana na NativeScriptu. Največja razlika po času je sicer na tej strani, ampak pri dejanski uporabi ni tako opazna, kot na uvodnem zaslonu. Najbolj konkurenčna je stran s podatki o oddaji, saj se v primerjavi z domorodno naloži še v isti ali naslednji sekundi. Zanimiva je predvsem primerjava domorodne aplikacije na obeh napravah, saj ni velike razlike v hitrosti delovanja. Nasprotno velja za NativeScript, ki zaradi uporabe navideznega stroja dodatno upočasni aplikacijo.

Aplikaciji se razlikujeta tudi po velikosti. Namestitveni arhiv APK (ang. Android Package Kit) pri domorodni aplikaciji zaseda 1,6 MB. Aplikacija

	Lenovo Vibe Shot Z90		**Lenovo A1000**	
	Domorodna	NativeScript	Domorodna	NativeScript
Uvodni zaslon	3,76	7,37	4,04	12,09
Seznam kanalov	5,10	3,05	7,18	5,41
Podatki o oddaji	2,06	2,59	2,28	3,92

Tabela 6.3: Učinkovitost nalaganja strani ob prvem zagonu aplikacije oziroma po izbrisu njenih podatkov. Podani časi so zapisani v sekundah.

	Lenovo Vibe Shot Z90		**Lenovo A1000**	
	Domorodna	NativeScript	Domorodna	NativeScript
Uvodni zaslon	3,23	7,18	3,76	11,70
Seznam kanalov	0,43	2,32	0,75	3,43
Podatki o oddaji	1,02	1,72	1,01	2,34

Tabela 6.4: Učinkovitost nalaganja strani ob prisotnosti aplikacijskih podatkov na napravi. Podani časi so zapisani v sekundah.

izdelana z NativeScriptom zahteva 15,5 MB. Podobno je pri zasedenem prostoru po namestitvi. Domorodna aplikacija zavzame 15,7 MB, od česar 150 KB po zagonu aplikacije porabijo njeni podatki. Aplikacija osnovana na NativeScriptu zavzame 41,4 MB, kjer 11,3 MB predstavljajo aplikacijski podatki po prvem zagonu. Ne glede na to, da domorodna aplikacija nima sklopa opomnikov, je razlika precejšnja.

Primerjava med obema aplikacijama kaže v prid domorodni. Kljub temu ne smemo zanemariti dejstva, da smo aplikacijo osnovano na NativeScriptu izdelali še za iOS na isti kodni osnovi. Slednja je iz vidika vmesnika povsem primerljiva z domorodno, prav tako je njena hitrost dopustna za občasno uporabo.

Poglavje 7

Zaključek

V diplomski nalogi smo podrobneje spoznali področje TV sporedov. Predstavili smo procesa priprave in prikaza sporedov. Prvi obravnava postopke pridobivanja, obdelave in priprave izhoda, ki so ključni za vsa nadaljnja dejanja. S procesom se ukvarja ponudnik TV sporedov, katerega cilj je pripraviti ažurne in pravilne sporede. Drugi zajema pridobivanje obdelanih podatkov in prikazovanje teh v primerni obliki preko različnih medijev. S procesom se ukvarja uporabnik sporedov, katerega cilj je predstaviti podatke gledalcem v okviru lastnih storitev.

S pomočjo spletnih tehnologij smo izdelali tri aplikacije, ki so namenjene podpori in delovanju procesov z opisanega področja. Spletna in namizna aplikacija podpirata proces priprave sporedov in sta namenjeni ponudniku podatkov. Mobilna aplikacija spada v končni del procesa prikaza sporedov in je namenjena končnim uporabnikom. Osredotočili smo se na odjemalski del aplikacij, ki smo ga razvili z dobro podprtimi ogrodji in skušali slediti najboljšim praksam, vzorcem in principom s področja tehnologij programske opreme.

Opisali smo zmožnosti uporabe spletnih tehnologij in trende, ki predstavljajo temelj napredka tovrstnih aplikacij. Uporabili smo jih v kontekstu realnega problema, v osnovi dobro poznanega vsakemu posamezniku informacijske družbe. Izpostavili smo odločitve o izbiri tehnologij in orodij, ki

pomembno vplivajo na nadaljnji razvojni proces. Namenjene so predvsem razijalcem, ki se srečujejo s podobnimi vprašanji in potrebujejo odgovore uokvirjene v dejanske implementacije.

7.1 Sklepne ugotovitve

Tekom razvoja aplikacij smo pridobili dodatne izkušnje za delo z opisanimi tehnologijami in dognali stanje, na katerem temeljijo naše ugotovitve. Te bomo v nadaljevanju predstavili za vsako vrsto aplikacije posebej, nato pa izpostavili možno izboljšavo celotnega projekta.

7.1.1 Spletna aplikacija

Enostranske aplikacije na osnovi JavaScripta predstavljajo novodoben pristop k razvoju spletnih aplikacij. Angular, med množico ogrodij za izdelavo SPA, s svojo zasnovo botruje temu in hkrati uvaja uporabo najboljših praks. Največja prednost Angularja je način strukturiranja aplikacijske logike in podpora lastnim platformam, ki niso odvisne ob brskalnika. Slabost so v času razvoja predstavljale zbirke komponent, ki še niso na ustreznem nivoju za splošno uporabo. Razlog je v tem, da ogrodje še ne rešuje nekaterih postavk, ki so ključnega pomena za razvoj platformno neodvisnih aplikacij. Angular je v osnovi namenjen poslovno usmerjenim aplikacijam in posledično vključuje vse tehnologije, ki so potrebne za razvoj na takšnem nivoju. Zaradi tega pri manjših aplikacijah prihaja do nezadovoljstva v povezavi z velikostjo distribuirane različice. Zelo priporočljiva je tudi izgradnja aplikacije z načinom prevajanja AOT.

7.1.2 Namizna aplikacija

Electron izpostavlja programski vmesnik, ki razvijalcu omogoča dostop do zmožnosti namizja, a ne omejuje načina pristopa k razvoju. Strukturiranje izvajalnega okolja in dobro razdelan model komunikacije sta ključnega po-

mena za preostale dele namizne aplikacije. Za aplikacije večjih razsežnosti in postopoma izboljšanih aplikacij, je priporočljiva uvedba tehnik primerljivih Angularju. Največja prednost ogrodja je nabor funkcionalnosti namizja, katere lahko izrabimo s pomočjo spletnih tehnologij na različnih platformah. Slabost predstavljajo APIji, preko katerih ne moremo neposredno spreminjati lastnosti gradnikov, kot je to mogoče z uporabo domorodnih tehnologij. Negativna lastnost je tudi velikost distribuiranega paketa, ki že za zelo enostavno aplikacijo zavzema vsaj toliko prostora, kot ga potrebuje izvajalno okolje ogrodja (približno 50 MB).

7.1.3 Mobilna aplikacija

NativeScript uvaja nov pristop k razvoju mobilnih aplikacij s pomočjo spletnih tehnologij. Največja prednost je možnost razvoja z jedrom ogrodja ali z integracijo z Angularjem, saj lahko izberemo tisto, ki nam bolj ustreza. V našem primeru smo se odločili za Angular predvsem iz strukturnega vidika ter možnosti prenosa njegove miselnosti v okvir mobilnih aplikacij. Posledično lahko ocenjujemo le slabosti, ki so povezane s tem načinom razvoja. Njegova največja težava je izris celotnega vmesnika po odprtju strani, saj ni v skladu s SPA. Omeniti je potrebno tudi nedelovanje nekaterih smernic v predlogah komponent iz jedra Angularja. Za pripravo distribuirane različice je priporočljiva uporaba načina prevajanja AOT, saj le v tem primeru aplikacija deluje tekoče. Razočarani smo bili nad učinkovitostjo aplikacije za Android. Trenutno uporabo ogrodja priporočamo v primeru, da se z njim primarno razvije aplikacijo za iOS ali enostavno aplikacijo za obe platformi.

Možna izboljšava v zvezi z razvojem in predstavitvijo zmožnosti, bi bila uporaba ene kodne osnove za vse aplikacije. V našem primeru smo to izvedli zgolj za spletno in namizno aplikacijo, saj mobilna aplikacija za podporo procesu priprave sporedov ni smiselna. Izziv v okviru ene kodne osnove bi predstavljalo strukturiranje projekta, kjer bi z enotno logično osnovo zadostili vsem načinom prikaza.

7.2 Spremna misel

Splet se je od izdelave preprostih statičnih strani precej spremenil. Tehnologije se razvijajo v obliko splošnonamenske uporabe in niso omejene na končno platformo. Posledično nastajajo ogrodja za izdelavo različnih vrst aplikacij. Čeprav pogled na spletne tehnologije mnogokrat ostaja v časih statičnih strani, se je od takrat z njihovim razvojem in množično uporabo, spremenil tudi način razvoja aplikacij. Naj bo to vodilo bralcu, v kolikor se prvič srečuje z razvojem predstavljenih vrst aplikacij.

Literatura

[1] Angular – Architecture Overview. URL: https://angular.io/guide/architecture. [Dostopano: 10.10.2017].

[2] Electron – Quick Start. URL: https://electron.atom.io/docs/tutorial/quick-start/. [Dostopano: 31.10.2017].

[3] NativeScript Documentation – Accessing Native APIs. URL: https://docs.nativescript.org/core-concepts/accessing-native-apis-with-javascript. [Dostopano: 14.11.2017].

[4] RxJS Manual – Overview. URL: http://reactivex.io/rxjs/manual/overview.html. [Dostopano: 9.10.2017].

[5] The definition of Interactive Program Guides and Electronic Program Guides. URL: http://www.itvdictionary.com/epg_ipg.html. [Dostopano: 6.10.2017].

[6] Tim Berners-Lee. The World Wide Web: A very short personal history. URL: https://www.w3.org/People/Berners-Lee/ShortHistory.htm, 1998. [Dostopano: 8.10.2017].

[7] Bert Bos. A brief history of CSS until 2016. URL: https://www.w3.org/Style/CSS20/history.html, 2016. [Dostopano: 8.10.2017].

[8] P. Cesar and K. Chorianopoulos. *The Evolution of TV Systems, Content, and Users Toward Interactivity.* Foundations and Trends(r) in Human-Computer Interaction. Now Publishers, 2009.

[9] Madhuri A Jadhav, Balkrishna R Sawant, and Anushree Deshmukh. Single Page Application using AngularJS. *International Journal of Computer Science and Information Technologies*, 6(3):2876–2879, 2015.

[10] Jens F Jensen. Interactive Television - A Brief Media History. In *European Conference on Interactive Television*, pages 1–10. Springer, 2008.

[11] K. Kawamoto. *Digital Journalism: Emerging Media and the Changing Horizons of Journalism*. Rowman & Littlefield, 2003.

[12] Michael S Mikowski and Josh C Powell. Single Page Web Applications. *B and W*, 2013.

[13] Cameron Nokes. Deep dive into Electron's main and renderer processes. URL: `https://medium.com/@ccnokes/deep-dive-into-electrons-main-and-renderer-processes-7a9599d5c9e2`, 2016. [Dostopano: 31.10.2017].

[14] Sebastián Peyrott. A Brief History of JavaScript. URL: `https://auth0.com/blog/a-brief-history-of-javascript/`, 2017. [Dostopano: 8.10.2017].

[15] Victor Savkin. Experiments with Angular Renderers. URL: `https://blog.nrwl.io/experiments-with-angular-renderers-c5f647d4fd9e`, 2016. [Dostopano: 10.10.2017].

[16] TJ VanToll. How NativeScript Works. URL: `https://developer.telerik.com/featured/nativescript-works/`, 2016. [Dostopano: 14.11.2017].