Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Černevšek

Medplatformski razvoj aplikacije za pregled in urejanje video posnetkov

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2014

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01979 / 2014
Datum: 30.1.2014

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GAŠPER ČERNEVŠEK**

Naslov: **MEDPLATFORMSKI RAZVOJ APLIKACIJE ZA PREGLED IN UREJANJE VIDEO POSNETKOV**
CROSS-PLATFORM APPLICATION DEVELOPMENT FOR VIDEO REVIEW AND EDITING

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Pri razvoju aplikacij je vedno bolj pomembna podpora različnim platformam, kar razvijalcem pogosto povzroča številne težave. Obstajajo različni pristopi reševanja teh težav - od razvoja ločenih aplikacij za posamezno platformo, do uporabe ogrodij, ki skrbijo za delovanje na več platformah. V okviru diplomske naloge naj študent razišče možnosti uporabe ogrodij Chrome Legacy Packaged Apps in Chrome Packaged Apps pri razvoju aplikacije za pregled in urejanje video posnetkov. Pri tem naj nadgradi uporabniško izkušnjo portala YouTube, kjer se naj osredotoči predvsem na izboljšanje nadzora nad pregledom naročenih kanalov. Pri tehničnem vidiku naj študent poskrbi za optimizacijo porabe procesorskega časa in spominskega prostora, kar je pri medplatformskem razvoju aplikacij zelo pomemben vidik.

Mentor:

doc. dr. Dejan Lavbič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Gašper Černevšek, z vpisno številko **63080003**, sem avtor diplomskega dela z naslovom:

Medplatformski razvoj aplikacije za pregled in urejanje video posnetkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 3. februarja 2014 Podpis avtorja:

Zahvalil bi se mentorju, doc. dr. Dejanu Lavbiču, za strokovno pomoč in usmerjanje med izdelavo diplomskega dela. Prav tako bi se zahvalil vsem ostalim, ki so mi stali ob strani vsa leta študija, predvsem svoji družini in punci Jani.

Kazalo

Kratice in njihovi pomeni

Povzetek

Abstract

1 Uvod **1**

2 Uporabljene tehnologije **5**

2.1 HTML5 in CSS3 5

2.2 JavaScript in jQuery 6

2.3 Chrome Legacy Packaged App 7

2.4 Chrome Packaged App 8

3 Razvoj aplikacije **11**

3.1 Razvojno okolje in uporabljena orodja 13

3.1.1 Razvojno okolje NetBeans 13

3.1.2 Adobe Photoshop 13

3.1.3 Sistem za nadzor različic TortoiseSVN 13

3.1.4 Brskalnik Google Chrome 14

Chrome DevTools 16

3.2 Različice aplikacije 17

3.3 Grafični uporabniški vmesnik 19

3.3.1 Osnovna postavitev 19

3.4 Uporabniške funkcionalnosti 24

3.5 Avtorizacija uporabnika 29
3.5.1 Osnovna konfiguracija 29
3.5.2 Realizacija avtorizacije v naši aplikaciji 31
3.6 Pridobivanje podatkov 32
3.6.1 YouTube Data Api v2 33
3.6.2 YouTube Data Api v3 35
3.6.3 Shranjevanje uporabniških podatkov 37
3.7 Struktura Packaged Appa 37
3.7.1 Datoteka manifest 38
3.7.2 Skripta v ozadju 40

4 Končna postavitev 43
4.1 Optimizacija slik 43
4.2 Optimizacija kode JavaScript 44
4.3 Optimizacija porabe spomina in procesorskega časa brskalnika 50
4.4 Sistem sporočil o napakah 54

5 Sklepne ugotovitve 55

Kratice in njihovi pomeni

AJAX (angl.) Asynchronous JavaScript and XML - skupina medsebojno povezanih spletnih razvojnih tehnik, uporabljenih za ustvarjanje interaktivnih spletnih aplikacij.
API (angl.) Application programming interface - definicija interakcije med posameznimi programskimi komponentami.
CSS (angl.) Cascading Style Sheets - preprosti slogovni jezik, ki skrbi za prezentacijo strani.
CWS (angl.) Chrome Web Store - spletna trgovina dodatkov za brskalnik Google Chrome.
DNS (angl.) Domain Name System - sistem, ki prevede uporabniku prijazna imena domen v številko IP.
DOM (angl.) Document Object Model - jezikovno neodvisna konvencija za prestavitev in interakcijo z objekti v dokumentih HTML.
HDMI (angl.) High-Definition Multimedia Interface - kompakten video/avdio vmesnik za prenos nekompresiranega zvoka in slike med napravami.
HTML (angl.) HyperText Markup Language - označevalni jezik za izdelavo spletnih strani.
JSON (angl.) JavaScript Object Notation - odprto standardni format, ki omogoča zapis objektov v človeško berljivi obliki.
JSONP (angl.) JSON with padding - tehnika, ki omogoča komunikacijo s strežniki na različnih domenskih naslovih.
JVM (angl.) Java Virtual Machine - platforma, ki omogoča izvajanje Javinih ukazov.

PDF (angl.) Portable Document Format - standard za izmenjavo elektronskih dokumentov, ki jih lahko prikažemo na vseh platformah.
PHP (angl.) Hypertext Preprocessor - odprtokodni programski jezik, ki se uporablja za razvoj dinamičnih spletnih vsebin.
REST (angl.) Representational state transfer - preprost način za pošiljanje in prejemanje podatkov med odjemalcem in strežnikom.
SVG (angl.) Scalable Vector Graphics - format za zapis dvodimenzionalne vektorske grafike s podporo za interakcijo in animacijo.
USB (angl.) Universal Serial Bus - večnamensko vodilo, namenjeno priklopu različnih perifernih naprav na računalnik.
VLQ (angl.) variable-length quantity - univerzalna koda za predstavitev poljubno velikega števila.
XML (angl.) Extensible Markup Language - računalniški jezik, ki omogoča format za opisovanje strukturnih podatkov.

Povzetek

V diplomskem delu je opisan medplatformski razvoj aplikacije za pregled in urejanje video posnetkov, ki jih uporabniki naložijo na priljubljeni spletni portal *YouTube*. Ta je z zadnjo večjo preobrazbo doživel veliko sprememb. Pojavil se je problem obveščanja uporabnikov o najnovejše naloženih video posnetkih, ki smo ga rešili z razvojem lastne aplikacije. Ta ponuja uporabniku veliko možnosti za prilagajanje njegovim željam, na primer lasten izbor kanalov. Vsak na novo naloženi posnetek je prikazan največ eno minuto po objavi, o čemer je uporabnik obveščen na več različnih načinov. Posnetke si lahko ogleda direktno v aplikaciji in si jih nato označi kot ogledane.

V diplomskem delu so opisane uporabljene spletne tehnologije, s pomočjo katerih je bila razvita aplikacija, ki deluje na platformah *Chrome Legacy Packaged Apps* in *Chrome Packaged Apps*. Razloženi so tudi postopki uporabe programskega vmesnika *YouTube Data Api v2* in *v3*, s katerimi lahko pridobivamo podatke o video posnetkih. Opisana je komunikacija za vzpostavitev avtorizacije uporabnika z njegovim Googlovim računom.

Med cilji diplomskega dela je bila tudi optimizacija porabe procesorskega časa in spominskega prostora, pri čemer smo uporabili orodje *Chrome DevTools* in *Closure Compiler*. Končni rezultat je delujoča medplatformska aplikacija *YouTube Video Deck*.

Ključne besede: medplatformski razvoj aplikacije, prikaz video posnetkov, uporabniška prilagodljivost, YouTube, Chrome Packaged Apps, Chrome DevTools, Closure Compiler

Abstract

This thesis describes the development of cross-platform application for over viewing and managing videos uploaded by users to popular website *YouTube*. In its latest major transformation many changes ocurred. One of the consequences was the problem of users not being alerted about the latest uploaded videos. Our application solves this problem. For users, it provides many options for adjusting their preferences, such as their own selection of channels. Each newly uploaded video is displayed in our application up to one minute after the video was published and users are informed about it in several different ways. Videos can also be viewed directly in the application and then marked as already seen.

This thesis describes web technologies that were used when developing our application which runs on *Chrome Legacy Packaged Apps* and *Chrome Packaged Apps* platforms. Communication to establish user authorization with his Google account is also described together with explained procedures for using *YouTube Data v2 & v3* API, that acquire information about videos.

Among the objectives of the thesis was also optimization of CPU time and memory usage. For this we used *Chrome DevTools* and *Closure Compiler*. The result is a functioning cross-platform application called *YouTube Video Deck*.

Keywords: cross-platform application development, video overview, user flexibility, YouTube, jQuery, Chrome Packaged Apps, Chrome DevTools, Closure Compiler

Poglavje 1

Uvod

Spletni portal YouTube je že vrsto let zelo priljubljena spletna stran. Glede na najnovejšo Googlovo statistiko vnesenih iskalnih pojmov v letu 2013, so uporabniki pojem *YouTube* iskali kar za 45 % večkrat kot v prejšnjih letih [11]. To potrjuje tudi trenutno tretje mesto na lestvici najpopularnejših spletnih strani, ki jo ureja podjetje Alexa [12]. Na portalu je že več kot 2 milijona in pol uporabnikov, ki svoje posnetke delijo z drugimi, mesečno pa portal obišče 1 bilijon unikatnih obiskovalcev. Uporabniki, ki delijo posnetke z drugimi, imajo na portalu svoj kanal, ki deluje kot izhodiščna točka za njihovo delo. Na kanalu so informacije o njihovih objavljenih posnetkih, aktivnostih in seznamih predvajanja. Uporabniki se lahko naročijo na kanal, katerega posnetki so jim všeč, in so ob vsakem novem posnetku lahko obveščeni po elektronski pošti v 10 do 30 minutah po objavi posnetka. Ti posnetki so bili pred prenovo spletnega portala vidni tudi na posebni uporabnikovi strani *Moje naročnine*, kjer so bili prikazani po vrsticah, pri čemer je vsaka vrstica prikazovala posamezni naročeni kanal. Ta pogled je uporabnikom olajšal pregled nad novimi posnetki, s čimer so ostali na tekočem z vsem, kar se dogaja z njihovimi priljubljenimi kanali.

Razvijalci portala YouTube se trudijo, da bi svojim uporabnikom predstavili čim več novih funkcionalnosti. Na vsake toliko časa pa portal doživi tudi malo večjo stilsko preobrazbo. Zadnja se je zgodila konec poletja 2012.

Ta preobrazba je prinesla veliko novosti in tudi slabosti. Celotna grafična podoba ni prilagojena zaslonom z večjo resolucijo, vsebina pa je poravnana levo in je fiksne širine, kar pomeni, da je na desni strani veliko neizkoriščenega belega prostora. Takrat je bila spremenjena tudi stran *Moje naročnine*. Posnetki zdaj niso več prikazani po vrsticah in združeni po kanalih, ampak so vsi v enem samem stolpcu in so med seboj pomešani. Sledenje naročenim kanalom je postalo oteženo, problem pa je bil še toliko večji zaradi občasnih napak, ko naročeni uporabniki niso bili obveščeni o novih objavljenih posnetkih. Zaradi tega se je velikokrat zgodilo, da so uporabniki spregledali posnetke, kar je imelo vpliv tudi na kanale YouTubea, saj je njihov zaslužek odvisen od števila gledalcev, ki si ogleda njihove posnetke.

Kot rešitev problemov na spletnem portalu YouTube je bila razvita aplikacija YouTube Video Deck. Ta uporabnikom omogoča, da so oni tisti, ki nadzirajo pregled nad naročenimi kanali. Po svojih željah lahko določijo, katere kanale bodo spremljali in na kakšen način. Omogoča jim, da so o novih objavljenih posnetkih obveščeni v največ 1 minuti po njihovi objavi. Obveščeni so lahko tudi s sistemskimi opozorili, ki se pojavijo, tudi če aplikacija ni aktivna. Aplikacija izkoristi ves razpoložljivi zaslonski prostor uporabnika, je enostavna za uporabo, pregledna, ponuja uporabniku tekočo interakcijo z vsebino ter vsebuje skoraj vse informacije in funkcionalnosti, ki so na spletnem portalu YouTube.

V 2. poglavju so opisane tehnologije, uporabljene pri razvoju aplikacije, v 3. poglavju pa so opisani vsi postopki razvoja aplikacije skupaj z uporabljenimi orodji. Podrobno so opisane vse funkcionalnosti skupaj z obrazložitvijo določenih oblikovnih odločitev in tudi sama avtorizacija z Googlovim računom uporabnika, ki je pomemben del aplikacije. Predstavljen je tudi način komunikacije s programskim vmesnikom na spletnem portalu YouTube. Ker je bilo pri sami aplikaciji veliko poudarka predvsem na odlični uporabniški izkušnji, je 4. poglavje namenjeno tehnikam, ki so bile uporabljene pri optimizaciji kode in ostalih delov aplikacije. Na koncu, v 5. poglavju, podamo sklepne ugotovitve in predloge za izboljšave oziroma na-

daljnji razvoj aplikacije.

Poglavje 2

Uporabljene tehnologije

Naša aplikacija bo uporabljala tehnologiji HTML5 in CSS3, ki bosta definirali njeno strukturo in oblikovanje. Za uporabniško interakcijo bo skrbel skriptni programski jezik JavaScript skupaj s knjižnico jQuery, ki nam olajša manipulacijo z drevesom DOM. Izbira podanih tehnologij nam omogoča, da aplikacijo z manjšimi popravki objavimo na treh različnih platformah: `Chrome Legacy Packaged App`, `Chrome Packaged App` in kot spletno stran. Vse platforme skupaj omogočajo uporabo naše aplikacije na različnih napravah in ob različnih priložnostih. Prvi dve platformi sta specifični za uporabo s spletnim brskalnikom Google Chrome. Chrome Legacy Packaged App omogoča uporabo aplikacije v samem brskalniku. Platforma Chrome Packaged App našo aplikacijo namesti kot namizno aplikacijo, medtem ko spletna stran omogoča uporabo naše aplikacije tudi z drugimi brskalniki.

2.1 HTML5 in CSS3

HTML5 je strukturni jezik za predstavitev vsebine na svetovnem spletu in spada med glavne tehnologije interneta. HTML5 je peta različica standarda HTML (ustvarjenega leta 1990 in standariziranega kot HTML 4 leta 1997). Nova različica HTML-ja vsebuje nove elemente, kot so `<audio>`, `<video>` in `<canvas>`, prav tako pa podpira skalibilno vektorsko grafiko (SVG), kar na-

domesti element `<object>`, ki je bil pred tem uporabljen za veliko različnih stvari. Ti elementi omogočajo, da je vključevanje video, audio in ostale grafične vsebine tako enostavno kot vključitev slike. S tem odpravimo potrebo po dodatkih, ki so bili potrebni doslej (Flash za video posnetke). Novi različici so bili dodani elementi, kot so: `<section>`, `<article>`, `<header>` in `<nav>`. Namenjeni so obogatitvi semantične vsebine, saj tako brskalniki in namenski programi lažje izluščijo pomembno vsebino s spletnih strani [1].

Za razliko od CSS2 podlog je CSS3 razdeljen na več samostojnih dokumentov poimenovanih `moduli`. Vsak modul doda nove funkcionalnosti ali nadgradi že prej definirane zmožnosti CSS2, s čimer ohranja kompatibilnost s starejšimi različicami CSS-ja. Prvi osnutki CSS3 so bili objavljeni junija leta 1999. Zaradi modularnosti ima vsak modul različno stanje in stabilnost. Z junijem 2012 je bilo objavljenih več kot petdeset različnih modulov, od katerih so bili štirje sprejeti kot uradno priporočilo. To so: modul za barve, modul za *Media Queries*, modul za imenski prostor in modul za poizvedbe nivoja 3.

2.2 JavaScript in jQuery

Za interakcijo z uporabnikom skrbi programski jezik JavaScript. To je objektni skriptni programski jezik, ki je implementiran kot del spletnega brskalnika, tako da lahko skripte na strani odjemalca sledijo dejanjem uporabnika in nadzirajo brskalnik, komunicirajo asinhrono in spreminjajo vsebino prikazane spletne strani. JavaScript se največkrat uporabi za nalaganje nove vsebine na spletno stran s pomočjo AJAX-a, brez ponovne osvežitve strani. Omogoča animacijo, premikanje in raztegovanje elementov na strani. Predvaja lahko video in avdio vsebino ter preverja pravilnost vnosnih polj pri pošiljanju podatkov na strežnik [3]. Ker je JavaScript edini jezik, ki ima podporo v vseh popularnejših brskalnikih brez namestitve raznih dodatkov, je zelo priljubljen jezik za različne knjižnice, ki poenostavljajo interakcijo z uporabnikom in spletno stranjo. Čeprav je JavaScript skriptni jezik z dinamičnimi objekti,

je v zadnjih letih prišlo do velikega zmogljivostnega napredka in se lahko primerja z ostalimi prevedenimi programskimi jeziki. JavaScript ni podprt samo za namizne računalnike, ampak tudi za vse mobilne naprave, zaradi česar je trenutno najpopularnejši spletni jezik [7].

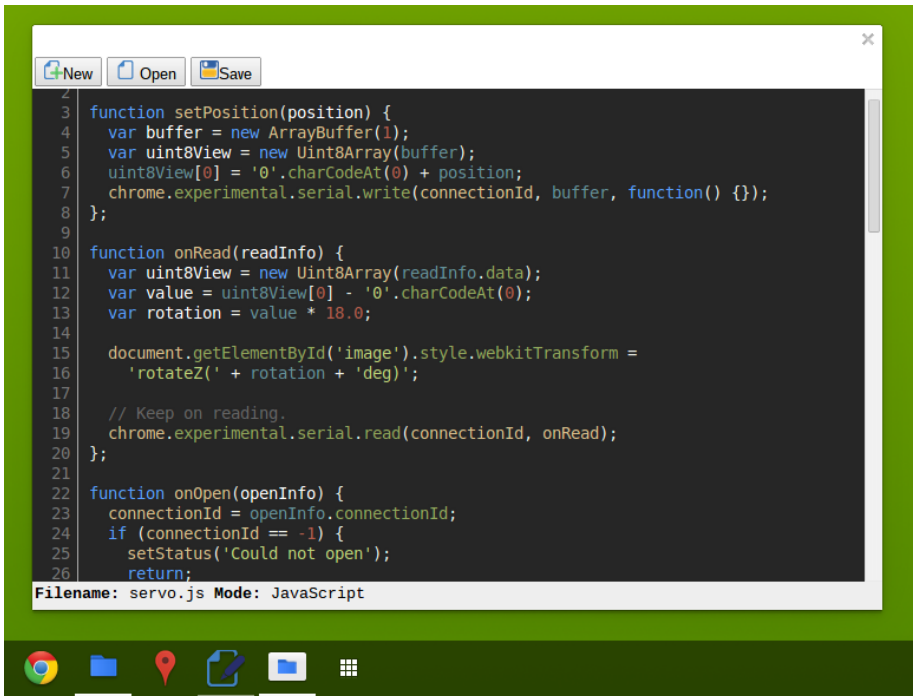
jQuery je knjižnica JavaScript s podporo za različne brskalnike, katere namen je poenostavitev interakcije s stranjo HTML [13]. Objavljena je bila januarja leta 2006, na seznamu 10.000 najbolj obiskanih spletnih strani pa jo najdemo v uporabi pri več kot 65 % straneh, s čimer ima status najpopularnejše knjižnice JavaScript [8].

2.3 Chrome Legacy Packaged App

Chrome Legacy Packaged App je bila prva različica Googlovih zapakiranih spletnih aplikacij [14]. Za razliko od obstoječih spletnih aplikacij je Packaged App aplikacija, ki je zapakirana v `.crx` datoteko in lahko uporablja zmogljivosti razširitev brskalnika Chrome. Packaged App lahko razdelimo na dva tipa. Prvi tip se imenuje *installable web app* in je spletna aplikacija, ki se v celoti namesti v brskalnik Chrome. Drugi tip se imenuje *hosted app*, je pa navadna spletna aplikacija z nekaj dodatnimi metapodatki. Oba tipa sta primerna za objavo v spletni trgovini Chrome. Le-ta je namenjena objavi razširitev in aplikacij za brskalnik Chrome in podpira različne načine za prezentacijo vsake aplikacije. Spletna trgovina vsebuje vse potrebne informacije o aplikaciji, opisuje, katere pravice potrebuje za svoje delovanje, in prikazuje, kako popularna je aplikacija. Uporabniki lahko komentirajo svojo izkušnjo z aplikacijo. Njihova mnenja so razvijalcem v veliko korist pri nadaljnjem razvoju te aplikacije. Ko uporabnik aplikacijo namesti v brskalnik, se mu le-ta prikaže kot ikona v za to namenjeni podstrani brskalnika. Omenjena stran služi nadzoru nameščenih aplikacij, ki jih uporabnik lahko zažene, odstrani ali doda bližnjice na namizje ali v orodno vrstico.

2.4 Chrome Packaged App

Chrome Packaged Apps ponujajo uporabniku enako uporabniško izkušnjo kot namizne aplikacije in so napisane v kombinaciji programskih jezikov HTML5, JavaScript in CSS [15]. Za razliko od spletnih aplikacij se obnašajo popolnoma enako kot namizne aplikacije in imajo dostop do sistemskih funkcij, Chrome API-jev in storitev, ki niso na voljo navadnim spletnim stranem. Tako lahko zgradimo aplikacijo, ki omogoča interakcijo z mrežnim prometom, strojno opremo, medijskimi orodji in drugimi komponentami računalnika. Ko uporabnik zažene Packaged App, je njegova pozornost usmerjena predvsem na opravila, ki jih aplikacija omogoča. Packaged App nima tradicionalne naslovne vrstice Chrome Omnibox in ostalih elementov, ki so prisotni v spletnih brskalnikih. Ob zagonu lahko aplikacija odpre okno, kot je prikazano na Sliki 2.1, in ga oblikuje na veliko različnih načinov.



Slika 2.1: Okno oblikovano v Chrome Packaged App.

Packaged App se vedno naloži lokalno, kar omogoča, da je aplikacija neodvisna od omrežja. Ko uporabnik aplikacijo namesti, ima v rokah popoln nadzor nad njenim življenjskim ciklom. Aplikacije se zaženejo in zaprejo hitro, in sistem lahko aplikacijo zaustavi kadarkoli, če je potrebna dodatna zmogljivost sistema. Brez razvijalčevega napora se bo aplikacija vedno zagnala, tudi ko omrežna povezava ni na voljo. Pri tem je treba poskrbeti, da so podatki ustrezno shranjeni v lokalni shrambi podatkov, dokler povezava ni na voljo, ko pa je povezava omogočena, se sinhronizirajo na naš podatkovni strežnik.

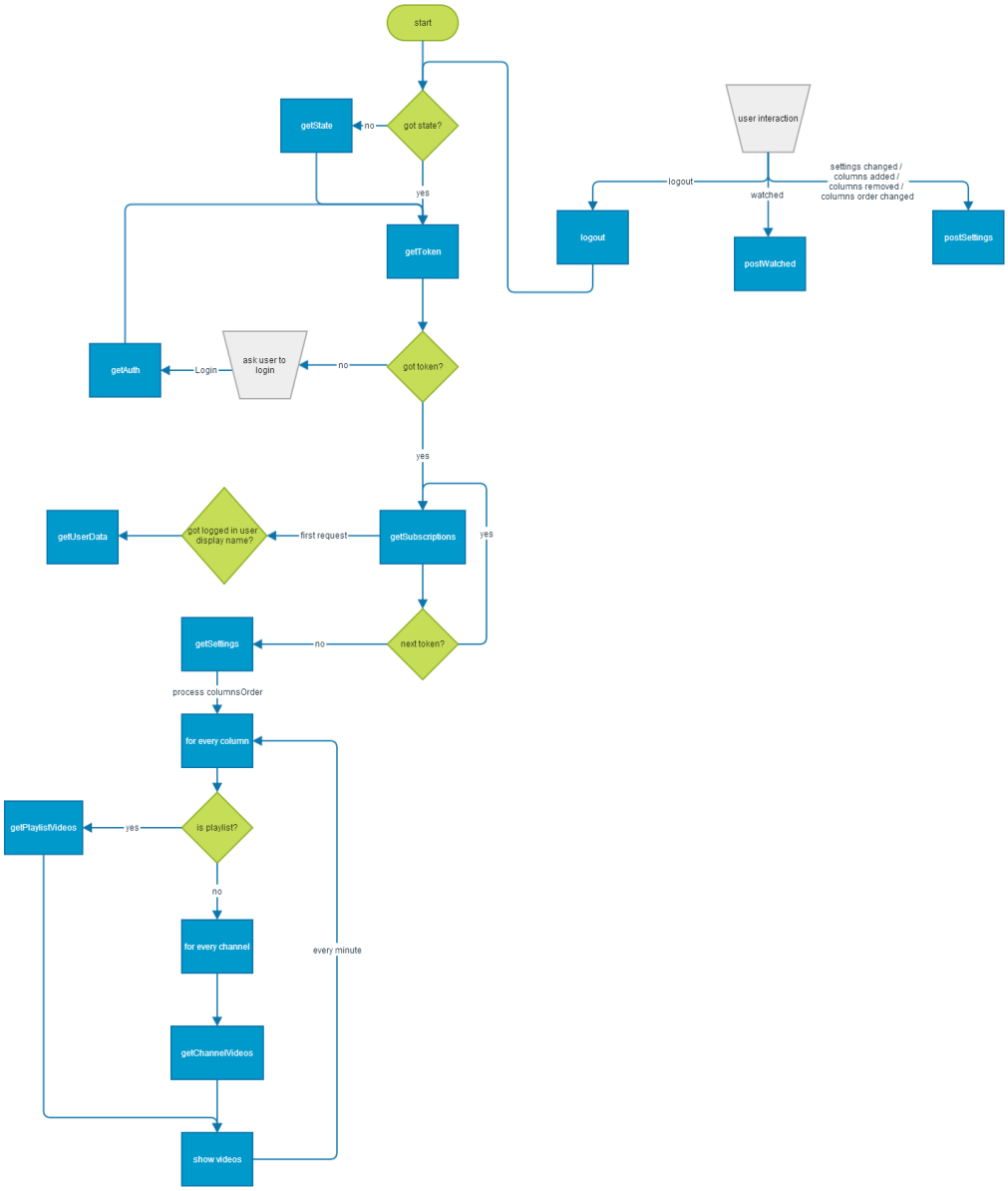
Pri razvoju aplikacije se uporabijo enaka koda, orodja in platforme kot za spletno platformo. Kljub temu so nekatere zmožnosti spletne platforme onemogočene oziroma uporabljene na drugačen način, s čimer se izognemo varnostnim vprašanjem in izboljšamo programerske prakse. Primeri onemogočenih funkcij so: `alert`, `confirm`, `document.cookie`, `document.close`, `document.open`, `document.write` [9].

Poglavje 3

Razvoj aplikacije

Razvoj aplikacije se je začel s spoznavanjem platforme Chrome Legacy Packaged App in njenih zmogljivosti. Razvijalci platforme so pripravili zadovoljivo dokumentacijo in primere uporabe, tako da je bila osnovna struktura aplikacije postavljena dokaj hitro skupaj s samim razvijalskim okoljem. Sledil je še pregled dokumentacije za komunikacijo s programskim vmesnikom spletnega portala YouTube. Tudi tu je bila ključnega pomena dokumentacija s praktičnimi primeri. Vzpostavitev komunikacije zato ni bila težavna, kar je pripeljalo do naslednjega koraka razvoja. Definirati je bilo treba logični potek klicev (slika 3.1), ki se bodo izvajali na naš strežnik in na programski vmesnik portala YouTube. Ključnega pomena pri tem je bilo pravilno zaporedje klicev za avtorizacijo uporabnika preko Googlovega računa [10]. Naslednji korak je bilo načrtovanje in oblikovanje vmesnika aplikacije, pri čemer je bil velik poudarek na sami uporabnosti aplikacije, saj je ravno to ključna prednost pred spletnim portalom YouTube. Idejna osnova oblikovanja temelji na priljubljeni aplikaciji TweetDeck [16]. Sledilo je sestavljanje posameznih delov v celoto, in rezultat je bila prva različica aplikacije. Ta je bila nato deležna še nekaj iteracij posodobitev funkcionalnosti in oblikovanja. Predzadnji korak je bil pregled možnosti optimizacije same aplikacije na različnih nivojih. Implementirane so bile izboljšave izvorne kode, prikaza grafičnih elementov, porabe procesorskega časa in sistemskega pomnilnika.

Za lažje odkrivanje napak v produkcijskem okolju pa je bil dodan tudi sistem za sporočanje napak. Na koncu so se v izvorni kodi naredile še ustrezne manjše spremembe za podporo aplikacije na ostalih dveh platformah.



Slika 3.1: Logični potek klicev.

3.1 Razvojno okolje in uporabljena orodja

3.1.1 Razvojno okolje NetBeans

NetBeans je integrirano razvojno okolje za primaren razvoj aplikacij Java, ki nudi podporo tudi za ostale jezike, kot so PHP, C/C++ in HTML5 skupaj z JavaScriptom [17]. NetBeans je spisan v programskem jeziku Java in lahko teče na vseh operacijskih sistemih, ki podpirajo Java Virtual Machine (JVM). Njegov razvoj se je začel leta 1996 kot študentski projekt pod imenom Xelfi. Leta 1997 je Roman Staněk ustanovil podjetje, ki je program ponudilo kot komercialno rešitev. Dve leti kasneje jih je kupilo podjetje Sun Microsystems in program NetBeans spremenilo v odprtokodno rešitev. Tako kot ostala razvojna okolja ima tudi omenjeni program orodja za profiliranje programske kode, grafično oblikovanje aplikacij in napreden urejevalnik JavaScript. Trenutno je javnosti na voljo različica 7.4.

3.1.2 Adobe Photoshop

Adobe Photoshop je vodilni profesionalni program na področju digitalnega urejanja slik [18]. Uporablja ga na milijone grafičnih umetnikov, tiskarskih mojstrov in navadnih uporabnikov po vsem svetu. Obstaja velika verjetnost, da je skoraj vsaka slika, ki ste jo videli obdelana oziroma ustvarjena s programom Photoshop. Uporablja se za izboljšavo in ustvarjanje novih slik, ki se lahko uporabijo v tisku in v digitalnem svetu. Ponuja veliko paleto različnih orodij za manipulacijo s sliko. Lahko jo povečamo, zmanjšamo, spremenimo osvetlitev, dodamo nove elemente na sliko, jih obarvamo na različne načine in s tem ustvarimo želeni rezultat.

3.1.3 Sistem za nadzor različic TortoiseSVN

TortoiseSVN je brezplačni program za razvijalce programske opreme [19]. Programerjem pomaga nadzirati veliko različic izvorne kode za njihovo aplikacijo. Omogoča, da lahko več razvijalcev istočasno dela na isti izvorni

kodi. Hrošči in nove funkcionalnosti so pogosto prisotne samo v določeni različici, zato je pomembno, da imamo možnost prehajanja med različnimi verzijami. Prav tako se lahko zgodi, da moramo razvijati dve različici programske opreme vzporedno. Primer tega je popravljanje hroščev in obenem razvijanje nove funkcionalnosti. Najenostavnejši način za početje česa takega je, da si za vsako različico naredimo kopijo programa in jih temu primerno poimenujemo. To pa zelo hitro postane neobvladljivo in časovno zahtevno za vzdrževanje, saj zahteva veliko samodiscipline razvijalcev in velikokrat pripelje do napak. Zato so bili razviti programi, ki ta postopek avtomatizirajo [4]. Primer takega programa je torej TortoiseSVN.

3.1.4 Brskalnik Google Chrome

Google Chrome je spletni brskalnik, ki ga razvija podjetje Google [20]. Prva poskusna različica je prišla na svetovni splet 2. septembra 2008 za Microsoft Windows. Skupaj s to različico je Google objavil tudi izvorno kodo skoraj kompletnega brskalnika kot odprtokodni projekt Chromium, na katerem temeljijo vse dosedanje različice. Med komponentami, ki niso bile objavljene pod odprto kodo, najdemo predvajalnik Flash in PDF-pregledovalnik. Do različice 27 je Google Chrome uporabljal spletni pogon WebKit, nato pa so ga nadomestili z vejitvijo WebKita, poimenovano Blink. Google Chrome cilja predvsem na varnost, hitrost, preprostost in stabilnost. Ena najmočnejših lastnosti je prav zmogljivost in hitrost procesiranja jezika JavaScript. Veliko unikatnih lastnosti brskalnika je bilo najprej najavljenih v ostalih konkurenčnih brskalnikih, vendar jih je Google Chrome prvi implementiral. Tak primer je grafično preoblikovanje naslovne vrstice (Omnibox), ki je bilo najprej najavljeno za brskalnik Firefox.

Brskalnik Google Chrome daje veliko prednost podpori standardov. Prva različica je prestala preizkusa Acid1 in Acid2, z različico 4 pa so prestali tudi test Acid3 [21]. Glede na rezultate preizkusa Test 262 ima Chrome zelo dobro podporo za JavaScript/ECMAScript, saj je brskalniku različice 28 med 11.583 testnimi primeri spodletelo samo pri 10 primerih [22]. Ta

dosežek ga uvršča med najboljše. Firefoxu 19 je – za primerjavo – spodletelo pri 193 poskusih, Internet Explorerju 9 pri več kot 600 poskusih, medtem ko je njegovemu nasledniku Internet Explorerju 10 spodletelo pri samo 7 poskusih. Prav tako je Google Chrome prestal 89.75 % preizkusov uradnega testa za CSS 2.1, ki ga vodi organizacija W3C. Med vodilnimi je tudi pri podpori standarda HTML5, saj je uspešno opravil 494 preizkusov od skupnih 555 [23].

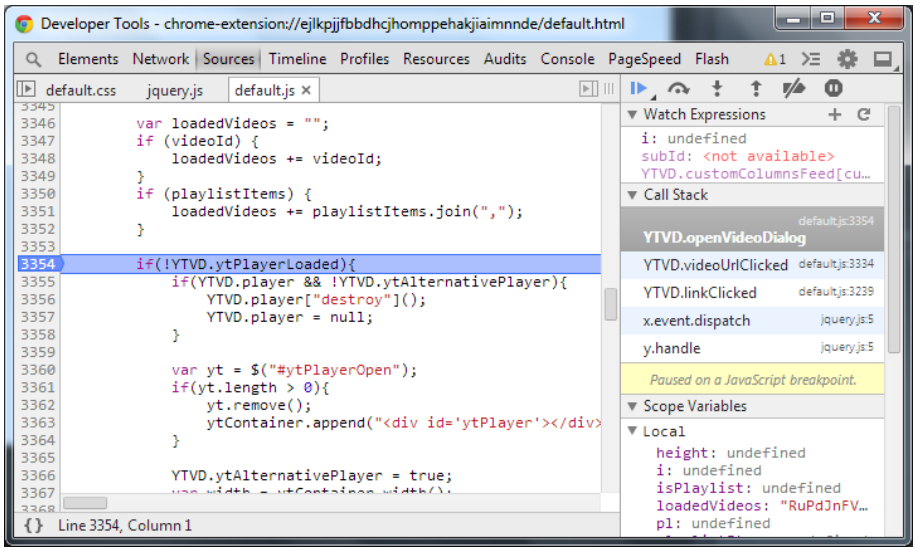
Hitrost je prav tako zelo pomembna vrlina brskalnika. Google Chrome uporablja pogon JavaScript pod imenom V8, ki podpira dinamično generacijo kode in precizno sproščanje pomnilnika. Na spletu najdemo veliko različnih orodij, s katerimi se ocenjuje zmogljivost pogona JavaScript, in tudi tukaj je Chrome v prednosti pred konkurenčnimi brskalniki. Poudarka pa niso dali samo na pogon JavaScript, ampak tudi na hitro odpiranje spletnih strani. Brskalnik ima vgrajen sistem za vnaprejšnje reševanje DNS poizvedb ter uporablja protokol SPDY, ki je hitrejši od protokola HTTP za komunikacijo s strežniki. Med stranmi, ki podpirajo protokol SPDY, najdemo Googlove storitve, Facebook, Twitter in druge večje spletne strani [24].

Kot ostali brskalniki tudi Google Chrome podpira razširitve. Obstaja več vrst razširitev. Razvijalci lahko z razširitvijo dodajo nove funkcionalnosti, spremenijo videz brskalnika ali pa ustvarijo celotno aplikacijo. Google je za razširitve pripravil temu namenjeno spletno trgovino, v kateri so objavljene brezplačne in plačljive razširitve. Leta 2011 je bilo objavljenih že več kot 11.500 razširitev [25].

Uporabniki so Google Chrome ob svojem prihodu toplo sprejeli, leta 2011 pa je po številu uporabnikov prehitel Firefox. Na konferenci `Google I/O` so Googlovi razvijalci objavili informacijo, da imajo 310 milijonov aktivnih uporabnikov, kar je več kot dvakrat toliko, kot so jih imeli leto poprej. Po zadnjih podatkih ima Google Chrome trenutno 43 odstotni svetovni delež uporabnikov.

Chrome DevTools

Chrome Developer Tools je nabor orodij za razhroščevanje in urejanje spletnih strani. Orodja so vgrajena direktno v brskalnik Google Chrome, kar omogoča razvijalcem dostop do internih lastnosti brskalnika. Nabor orodij je organiziran v skupine jezičkov, razporejenih po zgornji strani okna. Vsak jeziček je namenjen določenemu opravilu. Prvi jeziček je poimenovan *Elements*, v njem najdemo DOM-drevo trenutno odprte strani, omogoča pa nam interaktivno spreminjanje drevesa. Ob strani so prikazana tudi CSS-pravila izbranega elementa, kar olajša delo razvijalcem pri oblikovanju spletne strani ali pri preverjanju vrednosti elementov HTML. Jeziček *Console* je namenjen izpisu diagnostičnih podatkov, ki jih ustvarijo ukazi JavaScript, kot so `console.log` in `console.profile`. Prav tako ima razvijalec na voljo ukazno vrstico JavaScript, s katero lahko direktno upravlja spletno stran. Za pomoč pri razvoju kompleksnih aplikacij JavaScript si pomagamo z jezičkom *Sources*. Ta vsebuje vsa orodja za razhroščevanje aplikacije, vključno z zgodovino klicev, trenutnimi vrednostmi in polno podporo za prekinitvene točke.



Slika 3.2: Prikaz razhroščevanja z orodjem DevTools.

Aplikacije za delovanje pogosto potrebujejo povezavo s strežnikom za pre-

nos podatkov. V ta namen uporabljamo jeziček *Network*, v katerem so vidni vsi klici, ki jih brskalnik opravi na zunanje strežnike. Za vsak klic si je možno ogledati glavo zahtevkov in odgovorov skupaj s predogledom vrnjenih podatkov [6]. Jeziček *Audits* vsebuje orodja za testiranje in optimiziranje spletnih strani. Prikaže nam odvečna CSS- pravila ter nasvete za skrajšanje časa nalaganja spletne strani. Ker je za boljšo uporabniško izkušnjo potrebna tekoča interakcija s spletno stranjo, si pomagamo z jezičkom *Timeline*. V njem so prikazani vsi dogodki, ki se zgodijo v brskalniku. Za vsak dogodek imamo na voljo podatek, koliko časa se je izvajal. To nam pomaga pri odkrivanju ozkih grl v programski kodi. K sami zmogljivosti pripomore tudi poraba procesorskega časa in pomnilnika, kar lahko naredimo z orodji v jezičku *Profiles*. Z jezičkom *Resources* pa imamo nadzor nad vsemi podatki, ki so shranjeni v piškotkih in lokalni shrambi podatkov.

3.2 Različice aplikacije

Naša aplikacija je dostopna na treh različnih platformah: kot spletna stran, kot Legacy Packaged App in kot Packaged App. Vse tri platforme si delijo enako kodno osnovo, vendar ima vsaka svoje specifične lastnosti. Najprej je bila aplikacija razvita za platformo Legacy Packaged App. Za uporabo aplikacije na tej platformi jo morajo uporabniki najprej namestiti preko namenske trgovine Chrome Web Store [25]. To storijo tako, da aplikacijo najprej poiščejo med aplikacijami v trgovini, nato pa jo Google Chrome s klikom na `Namesti` gumb avtomatsko namesti v svoje okolje. V namenski podstrani z nameščenimi aplikacijami se doda ikona naše aplikacije. Ob kliku nanjo se aplikacija zažene v novem zavihku in se obnaša kot navadna spletna stran. Delo z aplikacijo končamo s tem, da zapremo zavihek. Ob desnem kliku na ikono aplikacije nam Google Chrome ponudi nekaj naprednejših možnosti. Uporabnik lahko izbira med različnimi načini zagona aplikacije: aplikacija se lahko vedno zažene kot navaden zavihek, lahko se zažene kot pripet zavihek, lahko se aplikacija odpre kot novo okno ali pa se zažene v celozaslon-

skem načinu. Uporabnik ima na voljo tudi možnost odstranitve aplikacije ali ustvarjanja bližnjic, s katerimi si olajša zagon aplikacije.

Naslednja stopnja v razvoju je bila podpora naše aplikacije kot spletne strani. Za razliko od prve platforme je bilo treba za splet narediti nekaj modifikacij v samem oblikovanju naše aplikacije. Google Chrome je med redkimi brskalniki, ki ponujajo možnost poljubnega oblikovanja drsnice. Ker je v našem oblikovanju drsnica pomemben strukturni element, je bilo treba za ostale priljubljene brskalnike najti alternativno metodo. Rešitev je bila knjižnica JavaScript `mCustomScrollBar`, ki posnema delovanje navadne drsnice, vendar ponuja možnost poljubnega oblikovanja. Slabost uporabe te rešitve je žrtvovanje nekaj procesorskega časa za enak videz aplikacije. Prav tako na spletu naletimo na težavo, kot je politika istega izvora, ki onemogoča brskalniku, da izvede zahtevke na strežnike, ki niso v istem domenskem prostoru, kot je trenutna spletna stran. Ta problem se rešuje z uvedbo komunikacije JSONP, ki obide politiko istega izvora z uporabo elementa HTML `<script>`. Temu elementu je dovoljena komunikacija do drugih domenskih prostorov, saj nam omogoča, da na našo spletno stran vključimo knjižnice JavaScript, ki niso naložene na našem strežniku. V JSONP se parametri zahtevka GET zapišejo kot dodatni parametri spletnega naslova zahtevka.

Zadnja podprta platforma je najnovejša iz serije Chrome Packaged Apps. Aplikaciji omogoča delovanje v čisto svojem okolju, v katerem se obnaša kot navadna namizna aplikacija. To pomeni, da mora aplikacija nadzirati delovanje okna, v katerem teče. Dodati je bilo treba osnovne gumbe za manipulacijo z aplikacijskimi okni, kot so: minimiziraj, maksimiraj in zapri. Prav tako je bilo treba določiti območje v navigacijskem delu aplikacije, ki podpira način premikanja aplikacijskega okna z miško. Za razliko od prejšnje različice platforme je pri tej varnost povečana. To pomeni, da morajo biti vsi zunanji viri preneseni v aplikacijo preko klicev. Dober primer zunanjih virov so slike. Te so v naši aplikaciji zelo pogoste, zato morajo biti v celoti prenesene najprej v delovni pomnilnik aplikacije in šele nato prikazane v sami aplikaciji. Zaradi varnosti prav tako ni mogoče v aplikacijo vključiti predvajalnika video

posnetkov na enak način kakor pri ostalih dveh platformah. Namesto uporabe predvajalnika Flash moramo uporabiti element `<webview>`, ki prav tako kot element `<iframe>` služi prikazu zunanjih spletnih strani, vendar vsebina elementa teče v drugem procesu, kar poveča varnost. Posebnost je tudi pri zunanjih storitvah, kot je Google Analytics. Za pravilno delovanje je potrebna uporaba HTML elementov `<iframe>` v peskovniškem načinu, pri čemer ima ta element beli seznam spletnih naslovov, do katerih ima dostop. Tako kot pri prvi različici platforme morajo uporabniki tudi tu aplikacijo najprej namestiti preko trgovine, potem pa jo lahko uporabljajo brez zagona brskalnika Google Chrome. Uporabniku je med drugim na voljo tudi razširitev – ki jo je pripravilo podjetje Google – ki v orodno vrstico operacijskega sistema doda bližnjice do vseh nameščenih aplikacij tipa Packaged App.

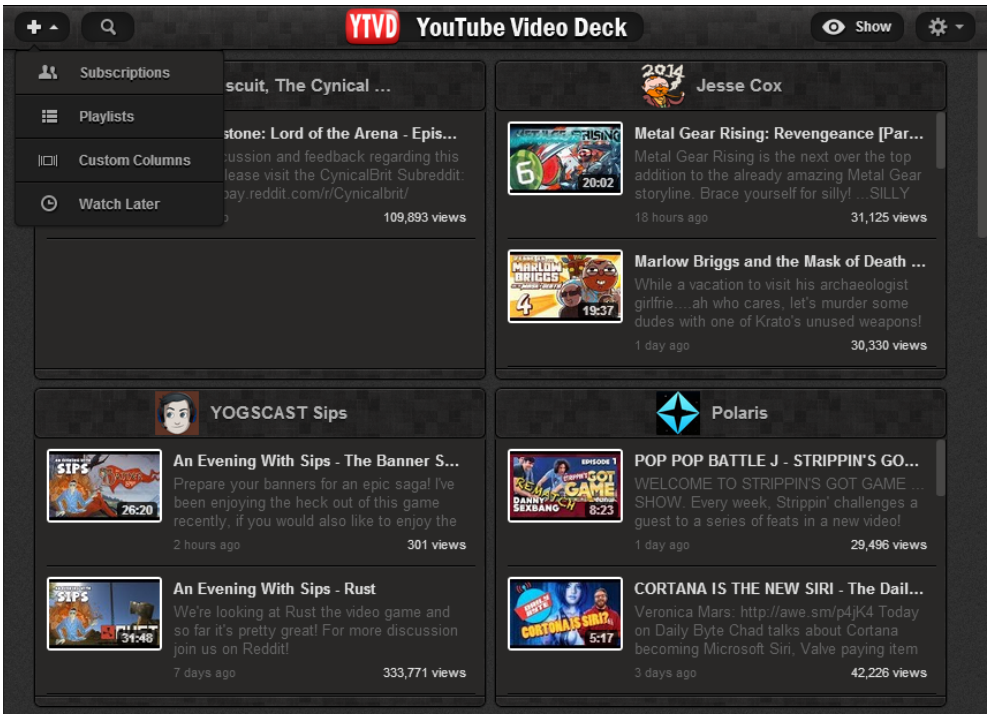
3.3 Grafični uporabniški vmesnik

3.3.1 Osnovna postavitev

Grafični vmesnik je razdeljen na dva dela. Na vrhu okna je naslovna vrstica, pod njo pa preostali del okna zaseda namizje (ang. deck). V naslovni vrstici so vsi gumbi za osnovno manipulacijo z objekti na namizju. Na levi je postavljen gumb za dodajanje objektov, ob kliku nanj pa se odpre podmeni, kjer ima uporabnik na izbiro, kaj želi dodati. Izbira lahko med naročninami, seznami predvajanja in stolpci po meri. Ob gumbu za dodajanje je stikalo, s katerim izberemo, ali želimo imeti v stolpcih na namizju prikazane že videne video posnetke ali ne. Sledi mu centriran logotip aplikacije, čisto na desni pa je gumb za interakcijo z nastavitvami namizja, upravljanje uporabniških podatkov ter možnost pošiljanja povratne informacije.

Osnovni gradnik namizja je stolpec. Razdeljen je na glavo in vsebino. V glavi so zapisani podatki o imenu stolpca skupaj z njegovo ikono in gumbi za upravljanje stolpca. Spreminjamo lahko vrstni red stolpca, ga izbrišemo, označimo vse video posnetke v stolpcu kot videne ali začnemo predvajanje vseh posnetkov v stolpcu. V stolpcu so prikazani posamezni posnetki. Vsak

posnetek je sestavljen iz naslova, kratkega opisa, avtorja posnetka, slike, števila ogledov, dolžine in pretečenega časa od objave posnetka. Ob prehodu miške čez posnetek se na njem prikažeta še dodatna gumba. Prvi se nahaja pod sliko posnetka in je namenjen za označevanje posnetka kot videnega. Če kliknemo nanj, celoten posnetek posivi, s čimer pozornost preusmeri na ostale, še ne videne posnetke. Drugi gumb je v desnem spodnjem kotu slike posnetka. Služi kot gumb za prikaz menija, v katerem se nahajajo dodatne možnosti, kot sta dodajanje posnetka v sezname predvajanja in deljenje posnetka na družbena omrežja. Če z miško več kot pol sekunde lebdimo nad posnetkom se začnejo na mestu slike prikazovati odseki iz posnetka. To služi uporabniku kot dodatna informacija oziroma hiter pregled vsebine video posnetka.



Slika 3.3: Izgled naše aplikacije.

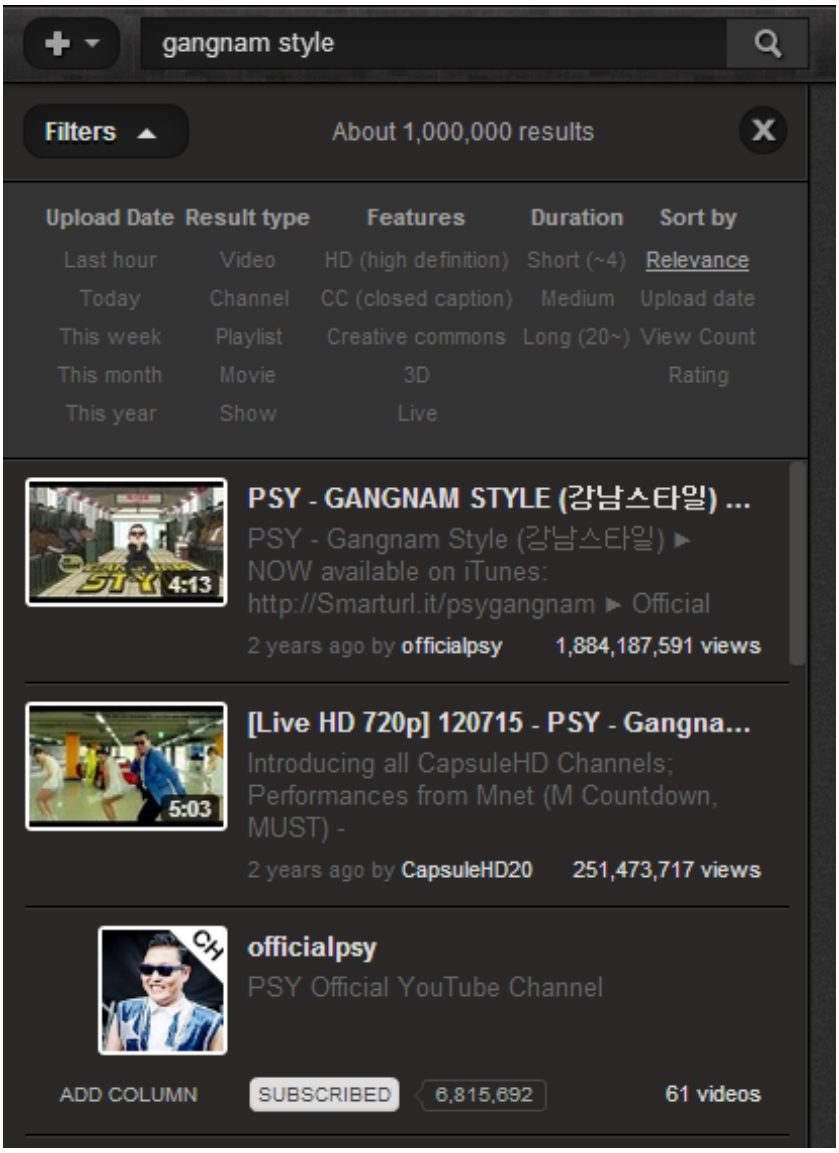
Ob kliku na posnetek se privzeto odpre predvajalnik video posnetkov. Realiziran je kot dialog, pri čemer se prilagodi razpoložljivi velikosti zaslona.

Glede na razmerje posnetka zapolni 95 % celotne širine ali višine zaslona. Ozadje dialoga se zatemni, kar da uporabniku kinematografsko izkušnjo in preusmeri vso pozornost na predvajani posnetek. Takoj ob odprtju predvajalnika se posnetek začne predvajati. V naslovni vrstici predvajalnika je nekaj gumbov in statusnih prikazov. Najprej je gumb za prikaz komentarjev video posnetka. Ob kliku nanj se dialog ustrezno razširi in levo od posnetka se prikažejo zadnji komentarji. Vsak komentar je sestavljen iz profilne slike avtorja komentarja, njegovega imena, časa objave in vsebine. Nad vsemi komentarji je prostor za objavo lastnega komentarja. Takoj pod tem poljem pa je možnost izbire vrstnega reda komentarjev. Desno od gumba za prikaz komentarjev je območje namenjeno prikazu splošnega mnenja o posnetku. Najprej sta prikazana gumba za izraz svojega pozitivnega ali negativnega mnenja, nato pa je še grafični prikaz trenutnega stanja mnenj skupaj z ustreznimi števili. Čisto na desni sta še dva gumba: prvi je namenjen skrčenju predvajalnika v spodnji desni kot zaslona, drugi pa zaprtju predvajalnika. Ob kliku na gumb za skrčitev predvajalnika se dialog preoblikuje. Ozadje postane ponovno vidno. Vse razen posnetka in desnih navigacijskih gumbov se odstrani. Ti trije elementi pa se prenesejo v spodnji desni kot zaslona. S tem dajo uporabniku možnost nemotenega spremljanja posnetka in upravljanja namizja aplikacije. Ob ponovnem kliku na gumb za skrčitev predvajalnika pa se ta povrne v prvotno stanje.

Slika 3.4: Predvajalnik video posnetkov.

Iskalni vmesnik je direktno v naslovni vrstici. Sestavljen je iz dveh delov. Na levi je najprej iskalno polje, na desni pa iskalni gumb. Ko uporabnik vnese iskalni niz v polje, lahko sproži iskanje s pritiskom na tipko enter ali s klikom na iskalni gumb. Po sprožitvi se z leve strani namizja prikažejo rezultati. Na vrhu rezultatov je gumb za izbiro iskalnih filtrov, desno od njega pa je prikaz števila zadetkov. Ob kliku na gumb za izbiro iskalnih filtrov se pod gumbom prikažejo različne kategorije filtrov skupaj z možnimi izbirami. Ob kliku na posamezno izbiro se iskanje ponovi z upoštevanjem izbranega filtra. Ko je izbran katerikoli od iskalnih filtrov, se prikaže v desnem spodnjem kotu gumb za ponastavitev izbranih filtrov. Prikazani zadetki so predstavljeni skoraj enako kot posnetki v stolpcih. Razlika je v različnih vrstah rezultatov. Zadetki namreč niso samo video posnetki, ampak tudi kanali YouTube in različni seznami predvajanja. Oboji so prav tako predstavljeni z naslovom, opisom, prikazno sliko, avtorjem in številom posnetkov. Tip zadetka je predstavljen z ustreznim trakom, ki prekriva prikazno sliko v zgornjem

desnem kotu. Prav tako je pod prikazno sliko gumb, s katerim dodamo kanal oziroma seznam predvajanja na namizje kot stolpec. V meniju dodatnih možnosti lahko rezultat dodamo v enega izmed *stolpcev po meri*. Pri zadetkih tipa kanal YouTube pa imamo dodan tudi gumb, s katerim se naročimo na ta kanal, skupaj s prikazom števila uporabnikov, ki so že naročeni.



Slika 3.5: Prikaz rezultatov iskanja.

3.4 Uporabniške funkcionalnosti

Glavna lastnost aplikacije je enostaven pregled posnetkov naročnin YouTube. Ko uporabnik prvič zažene aplikacijo, se mora najprej prijaviti s svojim Googlovim računom. Uporabniku se prikaže okno za prijavo, ki mu sledi okno za dodelitev pravic uporabe njegovih podatkov aplikaciji. Če je uporabnik lastnik različnih kanalov YouTube, mora na koncu izbrati še ustrezen kanal. Po uspešni prijavi uporabnik lahko začne uporabljati našo aplikacijo.

Naslednji korak je izbira prikazanih stolpcev. Uporabnik lahko izbira med tremi različnimi tipi stolpcev. Prvi tip stolpca je za vsako ločeno naročnino. Uporabnik iz seznama izbere svojo želeno naročnino in jo s klikom doda na namizje. V stolpcu se prikažejo posnetki, izbrane naročnine, urejeni po kronološkem vrstnem redu od najnovejšega do najstarejšega. Drugi tip stolpca so seznami predvajanja. Uporabniku se prav tako prikaže seznam njegovih naročnin, na izbiro pa dobi tudi svoj kanal. Ob kliku na katerokoli naročnino se prikaže nov seznam, v katerem so zapisani ustvarjeni seznami predvajanja izbrane naročnine. Če uporabnik izbere lasten kanal, dobi med drugim tudi poseben seznam predvajanja, poimenovan *Oglejte si pozneje*. V ta seznam predvajanja si uporabnik lahko dodaja video posnetke, ki jih želi pogledati v nekem kasnejšem obdobju. Med uporabniki spletnega portala je ta seznam predvajanja zelo popularen, zato je njemu posvečena pozornost tudi v naši aplikaciji. Tretji tip stolpca je stolpec po meri. Ta uporabniku ponuja možnost združevanja različnih naročnin v en sam stolpec, s čimer si lahko ustvari stolpce z različnimi kategorijami naročnin. Pri dodajanju stolpca po meri se uporabniku prikaže seznam že obstoječih stolpcev po meri in možnost ustvarjanja novega. Klik na obstoječ stolpec po meri ali ustvarjanje novega odpre novo okno, v katerem uporabnik izbere naročnine, ki jih želi združiti v isti stolpec. Tega nato še poimenuje in shrani. Med vrstami stolpcev ima posebno lastnost stolpec *All Subscriptions*, v katerem so združeni objavljeni posnetki vseh kanalov, na katere je uporabnik naročen. Z njim si lahko uporabnik pomaga pri pregledu posnetkov kanalov, ki jih še nima dodanih na namizje.

Uporabnik lahko prikaz stolpcev na namizju prilagodi svojim potrebam. Osnovni pogled ima stolpce postavljene drugega ob drugem, kar uporabnikom omogoča, da izkoristijo višino svojega monitorja za prikaz večje količine posnetkov v stolpcu. Uporabnik v tem primeru večjo količino stolpcev pregleduje s horizontalno drsnico. Ta pogled ima svoje slabosti. Če je število posnetkov v stolpcu majhno, je velik del stolpca prazen in bi bil lahko izkoriščen za prikaz posnetkov ostalih stolpcev. Problem odpravi vertikalni pogled. V njem so stolpci razporejeni v mrežo in uporabniku je ponujena nastavitev za določanje višine stolpca, s čimer izbere, koliko posnetkov želi videti v vsakem stolpcu. V povprečju to pomeni, da lahko na isti površini prikažemo dvakrat večje število stolpcev. Pri tem je pomemben vrstni red prikaza stolpcev, zato je uporabniku omogočeno spreminjanje vrstnega reda stolpcev z namenskimi gumbi ali s tehniko primi in spusti.

Vsak stolpec vsebuje posnetke, ki so definirani s tipom stolpca. Posnetki so razporejeni po kronološkem vrstnem redu. Ob začetku se v stolpec naloži največ petnajst posnetkov. Seznam le-teh pa se dopolnjuje, ko se uporabnik pomika navzdol po posnetkih. Največje možno število naloženih posnetkov je tisoč, kar je tudi maksimalna omejitev programskega vmesnika YouTube. Vsak posnetek je predstavljen s svojim naslovom, kratkim opisom, številom ogledov in časom objave. Če gre za stolpec po meri ali seznam predvajanja se ob posnetku izpiše tudi avtor le-tega. Uporabnik ima pri vsakem posnetku na voljo obogatene funkcije. Izbran posnetek lahko doda v seznam predvajanja *Oglejte si pozneje*. Prav tako lahko posnetek deli s svojimi prijatelji preko družbenih omrežij, kot so Facebook, Twitter, Google+, tumbl, Blogger, reddit, LinkedIn in pinterest.

Akcijo klika na posamezni posnetek je možno izbirati. Privzeto se bo ob kliku odprl predvajalnik video posnetkov v naši aplikaciji. Alternativna možnost pa je, da se ob kliku uporabnik preusmeri na izvorno stran YouTube posnetka. Ko si uporabnik posnetek ogleda, ima možnost, da ga označi kot videnega, s čimer si ureja svoj seznam še nevidenih posnetkov v vsakem stolpcu posebej. Za še lažji pregled nad temi posnetki pa ima uporabnik tudi

Slika 3.6: Prikaz dodatnih možnosti.

možnost, da videne posnetke preprosto skrije. Tako so v vsakem stolpcu vidni samo še nevideni posnetki. Če stolpec ne vsebuje nobenega še nevidenega posnetka, se privzeto skrči na četrtino prvotne širine, s čimer sprosti svoj neizkoriščeni prostor na namizju ostalim stolpcem. Tako ima uporabnik boljši pregled nad svojimi naročninami in njihovimi novimi posnetki. Ima pa tudi možnost, da prazne stolpce preprosto skrije, dokler ne vsebujejo vsaj enega posnetka.

Novim uporabnikom je na voljo tudi napredna možnost, da vse posnetke v stolpcu označijo kot videne, saj so si jih veliko večino že poprej ogledali na portalu YouTube. Skupaj s to možnostjo pa je uporabniku na voljo tudi gumb, s katerim se vsi posnetki v stolpcu naložijo v naš predvajalnik posnetkov in se po kronološkem vrstnem redu od najstarejšega do najnovejšega začnejo predvajati drug za drugim. V takem vrstnem redu se predvajajo zato, ker si večina posnetkov sledi kot epizode neke serije. Zato začnemo najprej gledati prvo epizodo in nadaljujemo z drugo, in ne obratno. Predvajalnik posnetkov se odpre kot celozaslonski dialog, pri čemer zatemni ozadje.

Tako ima uporabnik svojo pozornost usmerjeno samo v posnetek, in ne v stvari, ki se dogajajo v ozadju. Če ima željo po nadaljnjem delu na namizju in ne želi zapreti posnetka, pa ima na voljo možnost premika predvajalnika posnetka v desni spodnji kot aplikacije, pri čemer se posnetek skrči na širino enega stolpca, tako da ne ovira dela na namizju. Ker se privzeta kakovost predvajanega posnetka določi na podlagi hitrosti uporabnikove spletne povezave, lahko občasno pride do anomalij, kot so nezaželeni preskoki na nižjo oziroma višjo kakovost. Zato je uporabniku na voljo možnost nastavitve fiksne kakovosti, s čimer se izognemo možnim anomalijam.

V predvajalniku video posnetkov se uporabniku prikaže tudi informacija o sami priljubljenosti predvajanega posnetka. Prikazani sta števili, koliko gledalcem je bil in koliko jim posnetek ni bil všeč. Uporabnik lahko svoje mnenje izrazi s klikom na ustrezni gumb. Pomemben del vsakega posnetka pa niso le izražanja v številkah, ampak tudi sami komentarji. Zato imajo uporabniki na voljo prikaz komentarjev, ki jih lahko razvrščajo glede na njihovo relevantnost ali kronološki vrstni red objave. Imajo pa tudi možnost objave svojega komentarja na posnetek.

Ker je naša aplikacija namenjena rednemu spremljanju video posnetkov, je uporabnik o vsakem novem posnetku tudi vizualno obveščen. Če ima ob tem času na zaslonu ravno našo aplikacijo, se okoli stolpca, v katerem je bil dodan posnetek, naredi začasna obroba, s čimer opozori nase. Če pa je naša aplikacija nekje v ozadju, se uporabniku v spodnjem desnem kotu prikaže opozorilo o novem video posnetku. V opozorilu je informacija o naslovu posnetka in njegovem opisu ter začetna slika video posnetka. Če naša aplikacija teče na platformi Chrome Packaged Apps, pa se uporabniku prikaže tudi profilna slika avtorja posnetka ter malo večja začetna slika novega video posnetka. Ob kliku na opozorilo se uporabnik preusmeri v aplikacijo, v kateri se začne novo-naloženi posnetek predvajati.

Ena glavnih lastnosti spletnega portala YouTube je iskanje po video posnetkih. Ta funkcionalnost je ključnega pomena tudi v naši aplikaciji, saj lahko uporabniki s tem v celoti zamenjajo uporabo portala YouTube z našo

aplikacijo. Z iskanjem lahko uporabniki odkrivajo nove zanimive posnetke. Rezultate lahko filtrirajo glede na tip, dolžino, datum objave, relevantnost in dodatne lastnosti video posnetkov. Med rezultati se lahko pojavijo posnetki, kanali in seznami predvajanja. Na najdene kanale YouTube se je mogoče prijaviti ter jih nato dodati na namizje kot stolpce. Da je uporabniku iskanje olajšano, so mu na voljo tudi iskalna priporočila, video posnetki pa imajo dodan prikaz priljubljenosti med gledalci.

Za boljšo uporabniško izkušnjo pri gledanju video posnetkov poskrbi podpora za napravo z imenom *Chromecast* [26]. Ta je malenkost večja kot USB-ključek in jo priključimo na TV-sprejemnik s pomočjo vhoda HDMI. Naprava se napaja preko USB-vhoda na sprejemniku. Za komunikacijo potrebuje brezžično omrežje. Ko predvajamo posnetek v svoji aplikaciji, le-ta preko omrežja zazna *Chromecast* in ponudi uporabniku gumb za povezavo. Ob kliku nanj se aplikacija poveže z napravo in ji pošlje podatke o predvajanem posnetku. Naprava nato na televizijskem zaslonu začne predvajati naš video posnetek. To uporabniku omogoča, da v udobju svoje dnevne sobe s pomočjo naše aplikacije spremlja nove video posnetke ter si jih ogleda na svojem TV-sprejemniku z velikim zaslonom.

Slika 3.7: Priklop Chromecast-a na TV sprejemnik.

3.5 Avtorizacija uporabnika

Aplikacija črpa svoje podatke s spletnega portala YouTube. Če želimo pridobiti uporabniške podatke, se morajo naši uporabniki avtorizirati. Ker je YouTube že pred leti prevzel Google, moramo za to uporabiti njihovo avtorizacijo. Google je za razvijalce pripravil lastno knjižnico, ki se lahko vključi v aplikacijo. Avtorizacija je realizirana s protokolom OAuth 2.0. Ko uporabnik prvič dostopa do naše aplikacije, se sproži avtorizacijski proces OAuth 2.0, aplikacija pa uporabnika preusmeri na Googlov avtorizacijski strežnik. Pri tem aplikacija poda strežniku podatke, do katerih želi aplikacija imeti dostop. Uporabniku se ti podatki izpišejo na zaslon v berljivi obliki, tako da je točno seznanjen s tem, do česa bo aplikacija z njegovim dovoljenjem dobila dostop. Ko uporabnik sprejme zahtevo, strežnik pošlje aplikaciji žeton, s katerim ta opravi nadaljnje dostope do podatkov. Žeton je veljaven samo omejen čas, zato je treba poskrbeti za pravilno osveževanje le-tega.

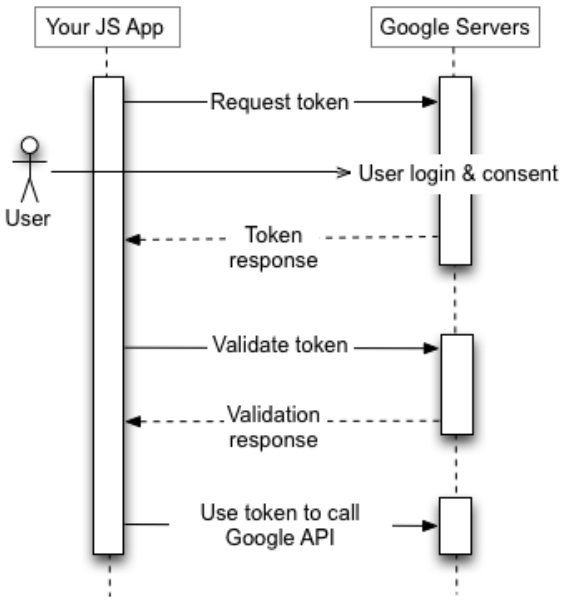
3.5.1 Osnovna konfiguracija

Vse aplikacije sledijo osnovnemu vzorcu pri dostopu do Googlovih programskih vmesnikov s pomočjo OAuth 2.0. Na visoki ravni lahko vzorec razdelimo na štiri osnovne korake.

Prvi korak je registracija naše aplikacije. Vse aplikacije, ki dostopajo do Googlovih programskih vmesnikov, morajo biti registrirane preko njihove Google Cloud konzole. S tem naša aplikacija pridobi ustrezne parametre, kot sta ID odjemalca in odjemalčeva skrivnost. Ti parametri so znani Googlu in naši aplikaciji. Prav tako so parametri odvisni od tipa aplikacije, ki jo razvijamo. Aplikacija JavaScript ne potrebuje odjemalčeve skrivnosti, medtem ko je za strežniške aplikacije potrebna.

Drugi korak je pridobitev žetona za dostop od Googlovega avtorizacijskega strežnika. Preden lahko naša aplikacija dostopa do zasebnih podatkov Googlovega programskega vmesnika, mora pridobiti žeton za dostop, ki dovoljuje dostop do posameznih klicev programskega vmesnika. En žeton

za dostop lahko dovoljuje različne nivoje dostopa do različnih programskih vmesnikov. Do katerih vmesnikov ima dostop, definiramo s spremenljivko `scope`. Zahtevo za žeton lahko realiziramo na več različnih načinov. Aplikacija JavaScript lahko zahteva žeton s preusmeritvijo brskalnika na Googlove strežnike, medtem ko naprave, ki nimajo brskalnika, zahtevo pošljejo direktno na spletno storitev za avtorizacijo. Nekatere zahteve zahtevajo še dodaten avtorizacijski korak, pri katerem se mora uporabnik prijaviti v svoj Googlov račun. Po uspešni prijavi se uporabniku prikaže stran, na kateri so napisana dovoljenja, ki jih zahteva naša aplikacija. Če uporabnik dovoli dostop, Googlov avtorizacijski strežnik pošlje aplikaciji žeton za dostop, v nasprotnem primeru pa javi napako.



Slika 3.8: Shema pridobitve žetona za dostop do Googlovih programskih vmesnikov.

Ko aplikacija pridobi žeton za dostop, ga lahko pošlje skupaj z zahtevkom na Googlov programski vmesnik v avtorizacijski glavi HTTP. Prav tako je mogoče poslati žeton kot parameter v spletnem naslovu zahtevka, vendar to

ni priporočljivo, ker lahko spletni naslovi zahtevkov končajo v nezavarovanih dnevnikih zahtevkov. Žetoni za dostop so veljavni samo za sklop dejavnosti in virov, ki so opisani v `scope` parametru zahtevka iz drugega koraka. Na primer: če je žeton za dostop uporabljen za dostop do programskega vmesnika YouTube, s tem aplikacija ne pridobi dostopa do Googlovega programskega vmesnika za koledar, vendar lahko večkrat uporabi isti žeton za dostop pri podobnih klicih na YouTube vmesnik.

Žetoni za dostop imajo omejeno časovno dobo. Če naša aplikacija potrebuje dostop do programskega vmesnika YouTube dalj časa, kakor je življenjska doba žetona za dostop, lahko pridobi žeton za osvežitev, s katerim lahko zahteva nove žetone za dostop. Ker so žetoni za osvežitev pomembni, jih je treba varno shraniti in uporabljati isti žeton, dokler je veljaven. Če zahtevamo večje število žetonov za osvežitev, kot je dovoljeno, bodo najstarejši žetoni postali neveljavni.

3.5.2 Realizacija avtorizacije v naši aplikaciji

Avtorizacija z Googlovimi strežniki je v naši aplikaciji realizirana s strežniško komunikacijo. Ker se na našem strežniku shranjujejo uporabniške nastavitve, mora naš strežnik imeti podatke o trenutno avtoriziranem uporabniku. Ob zagonu aplikacije le-ta preveri, ali je v njeni lokalni shrambi shranjen strežniški žeton. Če žetona še ni, ga zahteva s klicem na naš strežnik. Ta odgovori z žetonom, dolgim 200 znakov, ki predstavlja unikatno predstavitev aplikacije. Na strežniku se novo-generirani žeton shrani v bazo. Aplikacija strežniški žeton uporablja pri vseh nadaljnjih klicih na naš strežnik. Po pridobljenem strežniškem žetonu sledi zahtevek za dostopovni žeton. Na podlagi priloženega strežniškega žetona poskuša prepoznati uporabnika. Če žeton še ni povezan z določenim uporabnikom, strežnik odgovori aplikaciji, da se mora uporabnik najprej prijaviti. Aplikacija nato odpre pojavno okno, ki kaže na končno točko našega strežnika, ki je odgovorna za avtorizacijo. Strežnik uporabnika preusmeri na Googlov avtorizacijski strežnik, kjer se mora uporabnik prijaviti v njihov Googlov račun. Če gre za prvo uporabo aplikacije, se mu

pojavi tudi okno z opisom dovoljenj, ki jih zahteva aplikacija. Ko se uporabnik strinja z dovoljenji, Googlov avtorizacijski strežnik naredi preusmeritev nazaj na naš strežnik, pri čemer poda žeton za dostop in žeton za osvežitev. Slednji se zapiše v bazo skupaj z novim uporabnikom. Žeton za dostop pa se pošlje aplikaciji. Zdaj lahko aplikacija uporablja ta žeton za vse dostope do programskega vmesnika YouTube, vendar samo za omejen čas. Po preteku tridesetih minut mora na naš strežnik poslati zahtevo za osvežitev žetona za dostop. Strežnik uporabi shranjen žeton za osvežitev, ki ga pošlje skupaj z zahtevkom za nov dostopovni žeton na Googlov avtorizacijski strežnik. Ta odgovori z novim žetonom, ki ga naš strežnik posreduje aplikaciji za nadaljnje klice.

Taka implementacija omogoča uporabniku, da uporablja našo aplikacijo na različnih napravah. Pri tem vse podatke o uporabniškem računu hranijo Googlovi strežniki, s čimer je naš strežnik razbremenjen, ker mora hraniti samo podatke o uporabi in nastavitvah aplikacije. Prav tako nam podana rešitev omogoča enostavno implementacijo na različnih odjemalcih, saj se lahko pri uporabi drugačnih rešitev hitro pokažejo težave z namenskimi knjižnicami, ki so bile razvite za mobilne in drugačne naprave. S stališča analize uporabe naše aplikacije pa nam ponuja jasen pregled nad uporabniki, ker lahko pregledamo na koliko različnih napravah uporabljajo našo aplikacijo.

3.6 Pridobivanje podatkov

Podatki za aplikacijo so pridobljeni z uporabo storitve YouTube Data Api. Trenutno je na voljo že tretja različica storitve. Čeprav je označena kot stabilna, v njej ne najdemo vseh zmožnosti, ki jih ponuja storitev druge verzije. Vsaka ima svoje slabosti in prednosti, zato bomo za implementacijo svoje aplikacije uporabili kombinacijo obeh verzij storitev.

3.6.1 YouTube Data Api v2

Druga različica YouTube Data Apija je bila javnosti predstavljena leta 2008 in jo vse od takrat osvežujejo z novimi funkcionalnostmi, s čimer sledijo razvoju spletnega portala YouTube [27]. Storitev ponuja aplikaciji operacije, ki so na voljo portalu YouTube. Mogoče je iskanje po posnetkih ter pridobivanje podatkov o standardnih virih in sorodni vsebini. Z avtorizacijo uporabnika pa lahko nalagamo nove posnetke, spreminjamo uporabnikove sezname predvajanja in druge uporabniške interakcije. Storitev temelji na tehnologiji REST za branje, pisanje in spreminjanje podatkov. Omogoča dva različna načina dostopa. Prvi način je AtomPub: pri tem so podatki poslani kot objekti Atom. Drugi način je JSON: pri tem je struktura objekta enaka kot pri objektu Atom, vendar je zapisana v notaciji JSON. Vsaka funkcija je dostopna na svojem spletnem naslovu in ima definirane parametre, s katerimi dosežemo želen rezultat. Metoda zahtevka HTML določa, ali gre za branje, pisanje ali spreminjanje.

Primer 3.1: Zahteva za branje splošnih aktivnosti vseh naročnin

```
Host: gdata.youtube.com
Method: GET
Path: /feeds/api/users/default/subtivity?alt=json&max-results=25&start-index=1&v=2.1&orderby=published&inline=true
Authorization: Bearer ACCESS_TOKEN
```

Primer 3.2: Zahteva za dodajanje novega posnetka v seznam predvajanja *Oglejte si pozneje*

```
Host: gdata.youtube.com
Method: POST
Path:  /feeds/api/playlists/watch_later
Content-Type: application/atom+xml
Content-Length: CONTENT_LENGTH
Authorization: Bearer ACCESS_TOKEN
GData-Version: 2
X-GData-Key: key=DEVELOPER_KEY
Body:
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:yt="http://gdata.youtube.com/schemas/2007">
  <id>VIDEO_ID</id>
  <yt:position>1</yt:position>
</entry>
```

Primer 3.3: Zahteva za brisanje posnetka iz seznama predvajanja *Oglejte si pozneje*

```
Host: gdata.youtube.com
Method: DELETE
Path: /feeds/api/playlists/watch_later/PLAYLIST_ENTRY_
ID
Content-Type: application/atom+xml
Authorization: Bearer ACCESS_TOKEN
GData-Version: 2
X-GData-Key: key=DEVELOPER_KEY
```

3.6.2 YouTube Data Api v3

Tretja različica podatkovnega Apija je na voljo od začetka leta 2013 [28]. Glede na prejšnjo različico prinaša nekaj novejših tehnologij in drugačen koncept. Za celotno storitev je na voljo knjižnica JavaScript, ki jo razvijalci vključijo v svojo aplikacijo in nato za dostop do storitve kličejo ustrezno funkcijo. Ovrgli so metodo pošiljanja podatkov AtomPub, tako da zdaj uporabljajo izključno notacijo JSON. Dodali so podporo za gzip kompresijo poslanih in prejetih podatkov, kar zmanjša ustvarjeni promet med aplikacijo in storitvijo. Omogočili so uporabo ETags dela standarda HTTP. ETags je eden od mehanizmov, ki omogočajo validacijo za spletni predpomnilnik in pogojne zahteve na strežnike. To poveča zmogljivost predpomnilnika, saj lahko strežnik v primeru, da se vsebina odgovora ni spremenila, odgovori le delno, s čimer zmanjša količino pretečenih podatkov. Za razliko od druge verzije lahko aplikacija določi, katere parametre želi imeti v odgovoru storitve. S tem pa so razvijalce pozvali k optimalnosti njihovih rešitev z uvedbo omejitev klicev. Vsaka aplikacija ima dnevno na voljo končno število žetonov. Cena klica storitve je odvisna od podanih parametrov. Določeni parametri so ocenjeni kot dražji od drugih, zato je treba biti previden pri pogostem osveževanju podatkov v aplikaciji. Če gre za branje podatkov, prvi parameter stane 3 žetone, vsak naslednji pa še dodatna 2 žetona. Če gre za pisanje ali posodabljanje podatkov, prvi parameter stane 53 žetonov, vsak naslednji pa prav tako dodatna 2 žetona.

Primer 3.4: Zahteva za pridobitev informacij o uporabniškem kanalu

```
Host: content.googleapis.com
Method: GET
Path: /youtube/v3/channels?part=snippet%2CcontentDetai
ls&id= UCjOl2AUblVmg2rA_cRgZkFg
```

Primer 3.5: Odgovor na klic, ki je stal 5 žetonov


```
{ "kind": "youtube#channelListResponse",
 "etag": "\"KJzxKWJo5Mkivb-iOczuzoxz-Rk/qK-yb7M39qO_qZ
 dHq-be1870GWY\"",
 "pageInfo": {
  "totalResults": 1,
  "resultsPerPage": 5
 },
 "items": [{
   "kind": "youtube#channel",
   "etag": "\"KJzxKWJo5Mkivb-iOczuzoxz-Rk/ap3W386sJy1e
   yoiYy09-auDSxJs\"",
   "id": "UCjOl2AUblVmg2rA_cRgZkFg",
   "snippet": {
    "title": "TopGear",
    "description": "Want to watch a bit of Top Gear
       ...",
    "publishedAt": "2006-03-27T10:22:50.000Z",
    "thumbnails": {
     "default": {
      "url": "https://lh3.googleusercontent.com/-BDVUD
                SIn4XA/AAAAAAAAAAI/AAAAAAAAAAA/-4jOUGv
                IBdQ/s88-c-k-no/photo.jpg"
     }}},
   "contentDetails": {
    "relatedPlaylists": {
     "likes": "LLjOl2AUblVmg2rA_cRgZkFg",
     "uploads": "UUjOl2AUblVmg2rA_cRgZkFg"
    },
    "googlePlusUserId": "102309088840981609924"
   }}]}
```

3.6.3 Shranjevanje uporabniških podatkov

Ker bo aplikacija delovala na različnih platformah, je treba uporabniške nastavitve sinhronizirati med njimi. To storimo z lastnim podatkovnim strežnikom, kjer se shranijo nastavitve vmesnika in podatki o videnih video posnetkih. Komunikacija med aplikacijo in strežnikom je realizirana s HTML metodo POST, pri čemer se podatki pošljejo v notaciji JSON. Podatkovni strežnik podatke shranjuje v bazo MySQL. Avtorizacija podatkov se izvaja s pomočjo strežniškega žetona, na podlagi katerega pripiše podatke ustreznemu uporabniku. Vsak uporabnik ima lahko več teh žetonov, kar mu omogoča uporabo naše aplikacije na različnih napravah in platformah. Ker je poudarek pri razvoju naše aplikacije na uporabnosti in interaktivnosti, je pomembno, da se vsi klici na podatkovni strežnik izvedejo zelo hitro, ne glede na tip klica in velikost podatkov, ki jih mora strežnik vrniti. Pri tem veliko pripomore uporaba sistema *memcache*, ki omogoča shranjevanje rezultatov pogosto uporabljenih klicev na strežnik v pomnilniku strežnika. Ko se klic z določenimi parametri prvič zgodi na strežniku, *memcache* preveri, ali ima za prejet klic v pomnilniku že rezultat. Če rezultata v pomnilniku še ni, se klic posreduje dalje na MySQL. Ta odgovori na klic, rezultat pa se shrani v pomnilnik. Ob naslednjem klicu z istimi parametri se rezultat prebere direktno s pomnilnika, kar pripomore v povprečju k prepolovitvi izvajanja tega klica.

3.7 Struktura Packaged Appa

Aplikacija, razvita na platformi Chrome Packaged App, je zgrajena iz štirih glavnih komponent. Te so: `manifest` datoteka, skripta v ozadju, vsa koda aplikacije, ki vključuje HTML, CSS in JavaScript datoteke in ikone ter ostala sredstva, potrebna za delovanje aplikacije.

3.7.1 Datoteka manifest

Jedro aplikacije je datoteka `manifest.json`. Kot pove že končnica datoteke, je struktura le-te zapisana v notaciji JSON. V tej datoteki so definirani ime aplikacije, različica aplikacije, potrebna dovoljenja in ostali meta podatki, ki so potrebni za pravilen zagon in delovanje aplikacije. V naši aplikaciji je struktura datoteke `manifest` naslednja:

Primer 3.6: Definicija tipa aplikacije

```
"app": {
      "background": {
         "scripts": ["background.js"]
      }
   }
```

V tem segmentu definiramo, katera skripta se bo izvajala v ozadju in bo skrbela za življenjski cikel naše aplikacije. To je tudi segment, ki deklarira tip aplikacije. Če bi namesto `background` imeli strukturo `launch`, bi aplikacija tekla na platformi Legacy Packaged App.

Primer 3.7: Definicija ikon

```
"icons": {
      "128": "img/icon128.png",
      "16": "img/icon16.png",
      "24": "img/icon24.png",
      "32": "img/icon32.png",
      "48": "img/icon48.png"
}
```

Segment `icons` definira ikono naše aplikacije za različne velikosti. Google Chrome se na podlagi funkcionalnosti in velikosti razpoložljivega prostora odloči za pravilno ikono. Na primer: za ikono v jezičku bo uporabil velikost 16 x 16 px, na podstrani s seznamom nameščenih aplikacij pa bo uporabil ikono velikosti 128 x 128 px.

Primer 3.8: Definicija osnovnih podatkov

```
"description": "Best way to manage your YouTube
   subscriptions perfectly!",
"name": "YouTube Video Deck",
"minimum_chrome_version": "29",
"version": "0.9",
"manifest_version":2
```

Zgornji segmenti opisujejo osnovne podatke o aplikaciji. Posebnost je `minimum_chrome_version`, ki je nastavljen na 29 različico brskalnika Google Chrome zaradi potrebe po implementiranem programskem vmesniku `identity`, ki je bil predstavljen v tej različici.

Primer 3.9: Definicija dovoljenj

```
"permissions": [ "storage", "webview", "identity",
   "notifications", "unlimitedStorage",
   "https://*.ytimg.com/",
   "https://*.videodeck.net/",
   "http://gdata.youtube.com/",
   "https://apis.google.com/"]
```

Segment `permissions` je med najpomembnejšimi segmenti v datoteki `manifest`, saj v njem določimo, katere programske vmesnike bo naša aplikacija uporabljala in do katerih zunanjih strežnikov bo imela dostop. V naši aplikaciji nekatere podatke o uporabniku shranjujemo lokalno, zato potrebujemo dostop do vmesnika `storage`. Lokalna shramba podatkov je omejena na 5.242.880 bajtov (5 MB), kar pa lahko odpravimo z uporabo dovoljenja `unlimitedStorage`. Vmesnik `identity` se uporabi za avtorizacijo uporabnika preko našega strežnika, medtem ko se vmesnika `notifications` in `webview` uporabljata za obveščanje o najnovejših posnetkih in gledanju le-teh direktno v naši aplikaciji.

Primer 3.10: Definicija peskovnika

```
"sandbox": {
    "pages": ["ga/embedded_ga_host.html"]
}
```

V tem segmentu imamo definirano stran, ki bo tekla v peskovniškem načinu. Ta je potrebna za pravilno delovanje storitve Google Analytics.

3.7.2 Skripta v ozadju

Skripta v ozadju (ang. Background Script) je datoteka JavaScript, definirana v datoteki `manifest`. Ta skripta se zažene ob namestitvi aplikacije in je aktivna dokler imamo aplikacijo nameščeno. V njej so definirani dogodkovni poslušalci, ki se sprožijo ob zagonu, namestitvi, posodobitvi, mirovanju in odstranitvi aplikacije. Z njimi nadziramo našo aplikacijo in jo pravilno pripravimo glede na tip dogodka. Med vsemi dogodkovnimi poslušalci je nujno potreben samo tisti za zagon aplikacije. V naši aplikaciji ob tem dogodku odpremo novo okno, v katero naložimo aplikacijo.

Primer 3.11: Osnovna skripta v ozadju

```
chrome.app.runtime.onLaunched.addListener(function() {
    backend.openYTVD();
});

var backend = {
    openYTVD: function() {
        var settings = {
            "id": "YouTubeVideoDeckPackedApp",
            "frame": "none",
            "bounds": {"width": 1280, "height": 720,
               "top": 100, "left": 100},
            "minWidth": 580,
            "minHeight": 600
        };
        chrome.app.window.create('default.html',
           settings);
    },
}
```

Poglavje 4

Končna postavitev

Pomemben del vsake aplikacije je med drugim tudi, na kakšen način jo bomo zapakirali in objavili javnosti. Poudarek je predvsem na njeni zmogljivosti v spletnem brskalniku. Manj ko imamo zahtevkov na strežnik, hitreje se aplikacija naloži. Ker je aplikacija zapisana v skriptnem jeziku, je koda aplikacije golo besedilo, ki ga lahko bere vsak uporabnik naše aplikacije. S tem ogrozimo varnost svoje aplikacije in izvorne kode. Za lažjo detekcijo napak pa je priporočljivo vgraditi sistem za sporočanje napak aplikacije nazaj na strežnik, kar pripomore k hitrejšemu razhroščevanju aplikacije.

4.1 Optimizacija slik

Slike, ki so ustvarjene v programih, kot sta Photoshop in Illustrator, so videti odlične, vendar so velikosti datotek pogosto zelo velike. To nastane zaradi tega, ker so slike zapisane v formatu, ki omogoča lažjo manipulacijo s sliko na različne načine. Če naložimo na spletno stran slike velikosti tudi po nekaj megabajtov, se bodo le-te uporabniku prikazale zelo počasi. To pomeni, da moramo slike za spletno stran optimizirati glede na namenskost le-te. Slike vsebujejo podatke tudi o stvareh, ki niso le zaslonske pike. Ti podatki lahko sliko po nepotrebnem povečajo, kar privede do daljšega čakalnega časa, ko uporabnik čaka, da se slika prenese na računalnik. Optimiziranje teh podat-

kov lahko zmanjša velikost slike do 80 %. Poznamo dva tipa optimizacij, ki ju moramo razumeti. Prvi tip je izgubni. Slike, optimizirane s to metodo, bodo videti različne od originalne slike, vendar so razlike vidne samo ob podrobnem pogledu na originalno in optimizirano sliko. Velikost slike je temu primerno manjša. Drugi tip optimizacije slik je neizgubni. Pri njem je optimizirana slika identična originalu, vendar je njen zapis drugačen in je zato lahko manjši od originala. K hitrejšemu prenosu slike pripomore tudi njena ustrezna dimenzija. Če sliko, veliko več 1000 pik z uporabo oblikovanja CSS pomanjšamo na samo 20 pik, bo naš brskalnik s strežnika še vedno prenesel sliko v originalni velikosti. Zato je priporočljivo, da so slike na strežniku v enaki dimenziji, kot bodo prikazane na spletni strani. Velikokrat naletimo na spletne strani, katerih oblikovanje je sestavljeno iz velikega števila sličic. Pri odprtju tovrstne strani v brskalniku mora ta za vsako sliko poslati na strežnik zahtevo. Preveč hkratnih zahtevkov obremeni strežnik in poveča čas izvedbe zahtevka. Da to odpravimo, združimo več slik v eno, kar zmanjša število in skupno uporabo zahtevkov. Opisana tehnika se imenuje CSS Sprites [2].

Slika 4.1: Primer tehnike CSS Sprites.

4.2 Optimizacija kode JavaScript

Izvorna koda naše aplikacije je v celoti napisana v skriptnem jeziku JavaScript. To pomeni, da koda brez predhodne obdelave v aplikaciji vsebuje vse razvijalske komentarje, prazne znake in dolga imena funkcij in lastnosti. Rezultat je veliko nepotrebnih bajtov, ki so preneseni ob zagonu naše apli-

kacije. Problem rešujemo s postopkom minifikacije izvorne kode. Osnovni postopek odstrani nepotrebne dvojne presledke, nove vrstice in zamike. Za kompleksnejšo optimizacijo pa uporabimo orodja, kot so Closure Compiler [29], JSMin [30] ali YUI Compressor [31]. V naši aplikaciji je uporabljen Closure Compiler. Razvilo ga je podjetje Google v sklopu orodij Closure. Closure Compiler je orodje, ki omogoča hitrejši prenos in izvajanje kode JavaScript. Je pravi prevajalnik za JavaScript. Namesto prevajanja izvorne kode v strojni jezik prevede kodo JavaScript v *boljšo* kodo JavaScript. Izvorno kodo razčleni, analizira, odstrani mrtvo kodo ter prepiše in minimizira, kar je ostalo od prejšnjih korakov. Prav tako preverja sintakso, reference spremenljivk, tipe in opozori razvijalca na najpogostejše napake pri pisanju kode JavaScript.

Closure Compiler ima tri različne nivoje delovanja. Prvi se imenuje `whitespace_only`, in iz izvorne kode odstrani vse komentarje, prelome vrstic, nepotrebne presledke, nepotrebna ločila, kot so oklepaji in podpičja, in druge prazne znake. Rezultat je funkcionalna identična koda JavaScript, vendar ta nivo zagotavlja najmanjšo stopnjo kompresije. Drugi nivo se imenuje `simple_optimizations`, in opravi vse enako kot prvi nivo, vendar opravi tudi optimizacije znotraj izrazov in funkcij, kar vključuje preimenovanje lokalnih spremenljivk in parametrov funkcij v čim krajša imena. Preimenovanja v krajša imena naredijo kodo bistveno manjšo. Ker ta nivo optimizira samo lokalne spremenljivke in funkcije, to ne vpliva na interakcijo med optimiziranо in ostalo kodo JavaScript, zato je to tudi privzeti nivo optimizacije. `advanced_optimizations` je nivo tretje stopnje in naredi vse, kar naredi nivo druge stopnje, hkrati pa doda več agresivnejših transformacij, s čimer doseže najvišjo stopnjo kompresije. Ta nivo optimizira kodo veliko bolje, kot vsa ostala konkurenčna orodja. Za omogočitev tretjega nivoja naredi Closure Compiler močne predpostavke o naši kodi. Če koda ne ustreza tem predpostavkam, je rezultat ne-delujoča optimizirana koda. Na primer koda, optimizirana s tretjim nivojem, je lahko nekompatibilna z ne-optimizirano kodo, razen če storimo nekaj dodatnih korakov, ki zagotovijo kompatibilnost.

Če ne označimo imen zunanjih funkcij in lastnosti, jih bo Closure Compiler nepravilno preimenoval in s tem povzročil konflikte med našo in zunanjo kodo. Za razliko od drugega nivoja tretji nivo ne preimenuje samo lokalnih spremenljivk in funkcij, ampak tudi globalne. Prav tako odstrani kodo, ki je dokazano nedosegljiva. To je posebej uporabno v kombinaciji z velikimi knjižnicami. Če koda uporablja samo nekaj funkcij iz velike knjižnice, lahko Closure Compiler odstrani vse ostale funkcije razen uporabljenih. Ena od optimizacij pa je tudi tako imenovana `inlining`. Gre za optimizacijo, pri kateri Closure Compiler nadomesti klice funkcij s samim telesom funkcije. Ta metoda je uporabljena samo, ko prevajalnik ugotovi, da je optimizacija varna in prostorsko učinkovita. V nekaterih primerih se ta metoda uporabi tudi nad konstantami in nekaterimi drugimi vrednostmi.

Sledi primer kode, optimizirane na različnih nivojih.

Primer 4.1: Izvorna koda

```
/* To je neuporabljena funkcija */
function mrtvaFunkcija(sporocilo) {
   alert(sporocilo['text']);
}

function prikaziNaslovSporocila(sporocilo) {
   alert(sporocilo['naslov']);
}

var mojeSporocilo = {};
mojeSporocilo['naslov'] = "Moj opomnik";
prikaziNaslovSporocila(mojeSporocilo);
```

Primer 4.2: whitespace_only nivo

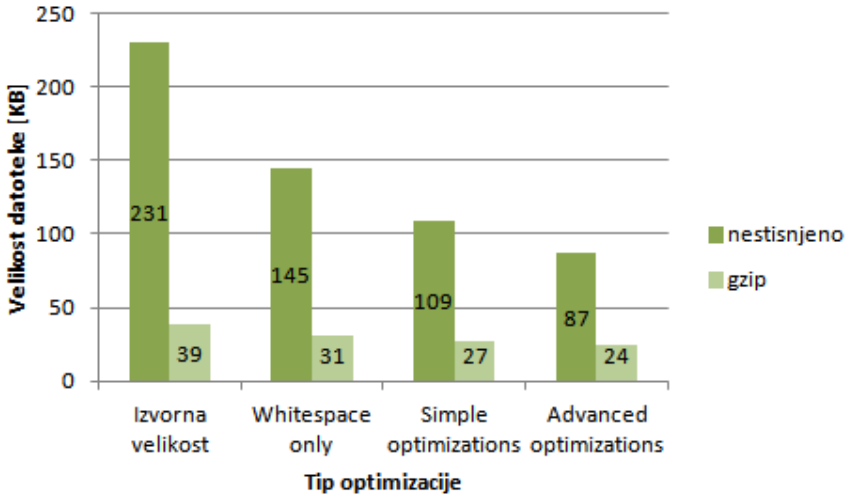
```
function mrtvaFunkcija(sporocilo){alert(sporocilo["tex
t"])}function prikaziNaslovSporocila(sporocilo){alert(
sporocilo["naslov"])}var mojeSporocilo={};mojeSporocil
o["naslov"]="Moj opomnik";prikaziNaslovSporocila(mojeS
porocilo);
```

Primer 4.3: simple_optimizations nivo

```
function mrtvaFunkcija(a){alert(a.text)}function prika
ziNaslovSporocila(a){alert(a.naslov)}var mojeSporocilo
={naslov:"Moj opomnik"};prikaziNaslovSporocila(mojeSpo
rocilo);
```

Primer 4.4: advanced_optimizations nivo

```
alert("Moj opomnik");
```



Slika 4.2: Primerjava velikosti optimizirane kode.

V svoji aplikaciji uporabljamo tretji nivo optimizacije. Zato morajo biti pri razvoju aplikacije upoštevana določena pravila pisanja kode. Najpomembnejše pravilo je, da so vse lastnosti objektov preimenovane, zato moramo biti pozorni, kako lastnosti definiramo in se kasneje nanje sklicujemo. Prav tako pomembno pravilo je med drugim tudi, da ne uporabljamo izraza `this`, ki se nanaša na kontekst izvajanja, le-ta pa se lahko po optimizaciji spremeni.

Primer 4.5: Različni načini dostopanja do lastnosti

```
var x = { renamed_property: 1 };
var y = x.renamed_property; // To je pravilno.

// Po preimenovanju 'renamed_property' ne obstaja na
   elementu x
//  zato naslednji pogoj vrne false.
if ( 'renamed_property' in x ) {}; // Nepravilno

// Tudi to je nepravilen sklic na vrednost:
x['renamed_property']; // Nepravilno
```

Če želimo ohraniti poimenovanje vrednosti, jo moramo definirati kot niz z uporabo narekovajev.

Primer 4.6: Ohranjanje poimenovanja lastnosti

```
var x = { 'unrenamed_property': 1 };
x['unrenamed_property'];  // To je pravilno.
if ( 'unrenamed_property' in x ) {};   // To je
   pravilno.
```

Naša aplikacija je razdeljena na dve datoteki JavaScript. V eni je koda za skripto v ozadju, v drugi pa ostala koda aplikacije. Ker mora biti med obema kodama povezava, moramo za to pravilno nastaviti zunanja imena funkcij in lastnosti. To je treba storiti za obe datoteki.

Primer 4.7: Definicija zunanjih imen funkcij in lastnosti

```
var backend = {
        YTVD: null,
        setIntervals: function() {},
        startPolling: function() {},
        createNotification: function(arg1) {}
};
```

Optimizirana koda nam prinese veliko prednosti, vendar ima tudi nekaj slabosti. Med največjimi je ta, da je koda zelo neberljiva in je zaradi tega razhroščevanje zelo oteženo. To pomanjkljivost odpravimo z uporabo datoteke source map, ki jo ustvari Closure Compiler. V osnovi ta datoteka definira preslikavo optimizirane kode nazaj v izvorno kodo. Če imamo podatek o vrstici in stolpcu, v katerem se je zgodila napaka, v optimizirani kodi lahko naredimo poizvedbo s pomočjo datoteke source map, ki vrne pozicijo v izvorni kodi. Podpora za datoteke source map je vgrajena direktno v Chrome DevTools ter tudi v brskalnik Firefox in omogoča, da vidimo izvorno kodo, čeprav je v resnici naložena optimizirana koda. Če želimo omogočiti avtomatsko uporabo datotek source map, moramo na konec optimizirane kode prilepiti naslednjo vrstico, ki vsebuje pot do datoteke source map:

Primer 4.8: Dekleracija source map datoteke v izvorni kodi

```
//# sourceMappingURL=/path/to/file.js.map
```

Lahko pa pot definiramo tudi v glavi zahtevka, ko vračamo vsebino datoteke JavaScript:

Primer 4.9: Dekleracija source map datoteke v glavi zahtevka

```
X-SourceMap: /path/to/file.js.map
```

Datoteka source map je zgrajena z notacijo JSON in vsebuje podatke, kot so različica izvorne kode, ime optimizirane datoteke, imena datotek, ki so bile združene v optimizirano kodo, imena vseh spremenljivk in funkcij, ki se

pojavijo v kodi, in nazadnje preimenovana imena, zapisana v formatu Base64 VLQ (Variable Length Quantity). Ta format je uporabljen zaradi tega, ker so bile datoteke source map najprej vsaj desetkrat večje od optimizirane kode. V tem formatu pa je datoteka source map večja samo še za faktor 1,5.

Primer 4.10: Datoteka source map

```
{
    version : 3,
    file: "out.js",
    sourceRoot : "",
    sources: ["foo.js", "bar.js"],
    names: ["src", "maps", "are", "fun"],
    mappings: "AAgBC,SAAQ,CAAEA"
}
```

V svoji aplikaciji uporabljamo datoteko source map skupaj s sistemom za sporočanje napak. Datoteka ni nikoli naložena pri samih uporabnikih, saj bi s tem razkrili izvorno kodo naše aplikacije. Shranjena je na strežniku, kjer se ob vsaki sporočeni napaki na podlagi različice aplikacije določi ustrezna datoteka source map, nad katero se nato naredi nekaj poizvedb, s čimer preslikamo napako v izvorno kodo.

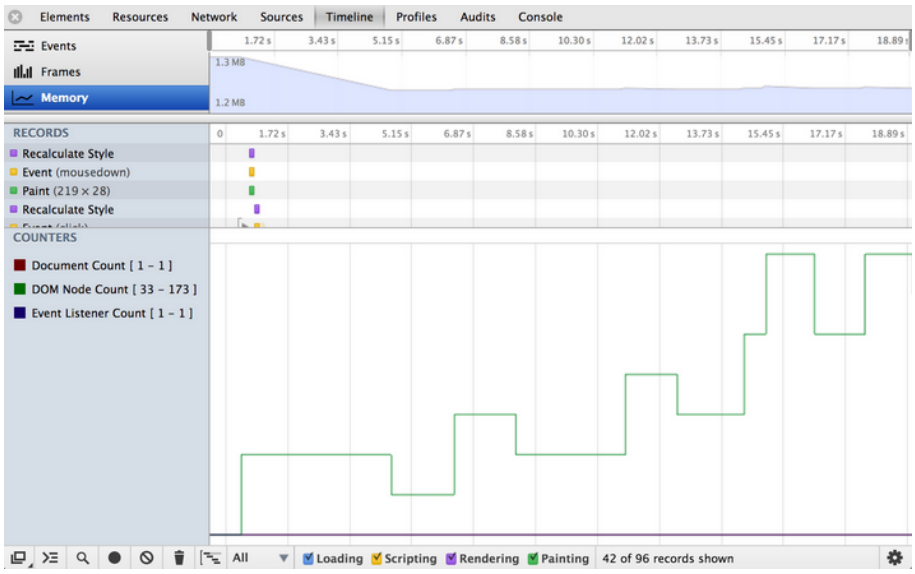
4.3 Optimizacija porabe spomina in procesorskega časa brskalnika

Puščanje pomnilnika je postopna izguba razpoložljivega pomnilnika računalnika. Do tega pride, ko aplikaciji večkrat ne uspe dostopati do pomnilniškega prostora, ki je bil namenjen začasni uporabi. Aplikacije JavaScript lahko pogosto trpijo zaradi podobnih težav, s katerimi se soočajo namizne aplikacije, kot sta puščanje in napihovanje pomnilnika, prav tako pa morajo upoštevati premore pri sproščanju pomnilnika. Čeprav aplikacije JavaScript uporabljajo

avtomatsko sproščanje pomnilnika, je učinkovito upravljanje pomnilnika še vedno potrebno [5].

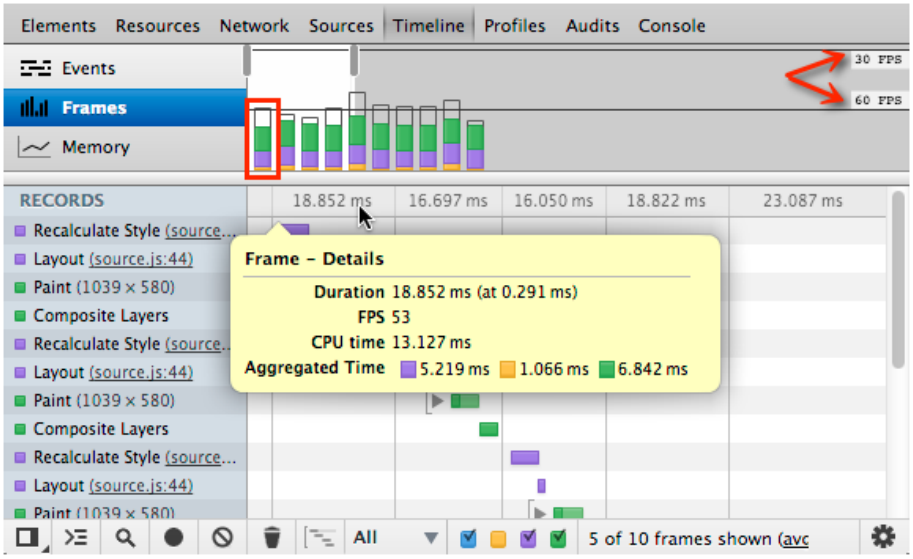
Prvi korak pri reševanju performančnih problemov je dokaz, da problem obstaja. To pomeni, da moramo najprej ustvariti ponovljiv test za merjenje osnovnega stanja. Brez takega testa ne moremo zanesljivo meriti svojega problema, prav tako pa ne moremo določiti, kdaj smo naredili s spremembo izboljšavo in kdaj smo nazadovali. Pri tem uporabimo Chrome DevTools, natančneje jeziček `Timeline`. Ta je zelo koristno orodje za prepoznavo našega problema. Omogoča popoln pregled, kje se porabi čas pri nalaganju in interakciji z našo aplikacijo. Vsi dogodki, od nalaganja virov, računanja prekrivnih slogov in premorov sproščanja pomnilnika do ponovnih izrisov, so narisani na časovnici. Pri preiskovanju pomnilniških težav lahko s pomočjo pogleda `Memory`, v jezičku `Timeline`, sledimo skupnemu dodeljenemu pomnilniškemu prostoru, številu vozlišč DOM, številu dokumentov in številu alociranih dogodkovnih poslušalcev.

Na sliki 4.3 vidimo tipični vzorec puščanja pomnilnika, pri katerem niso vsa vozlišča zbrana pri sproščanju pomnilnika.



Slika 4.3: Timeline jeziček v Chrome DevTools.

Sproščevalnik pomnilnika, kot je tisti v pogonu JavaScript V8, mora biti sposoben najti objekte v naši aplikaciji, ki so še vedno živi, prav tako pa tudi tiste, ki veljajo za mrtve in so zato nedostopni. Če sproščevalnik spregleda mrtve objekte zaradi logičnih napak v naši aplikaciji, potem spomin, ki ga zasedajo ti objekti, ne more biti več sproščen. To lahko povzroči, da sčasoma naša aplikacija postaja vse počasnejša. Do tega pogosto pride, ko napišemo programsko kodo na tak način, da so spremenljivke in poslušalci dogodkov, ki jih ne potrebujemo več, še vedno uporabljeni v živem delu kode. Dokler so ti sklici na mrtve objekte aktivni, jih sproščevalnik spomina ne more sprostiti. Priporočljivo je preveriti in izničiti spremenljivke, ki vsebujejo sklicevanja na elemente DOM, ki bi lahko bili spremenjeni oziroma uničeni med delovanjem aplikacije. Prav tako je treba preveriti lastnosti objektov, ki se lahko sklicujejo na druge objekte.

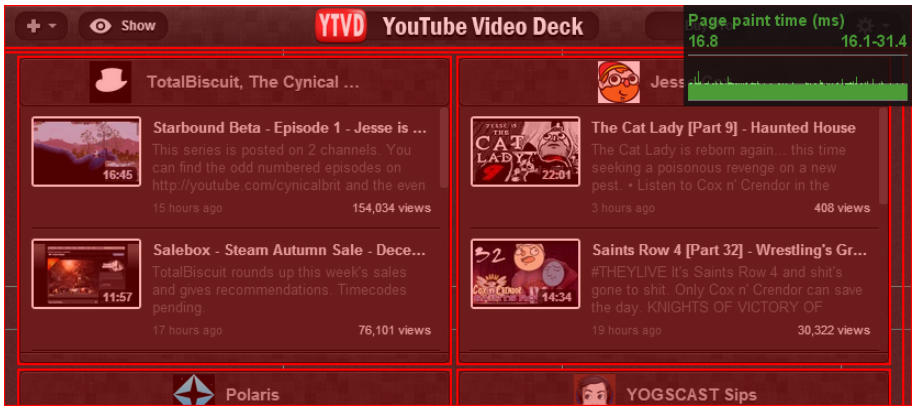


Slika 4.4: Prikaz izrisa strani v Chrome DevTools.

Ob razvoju svoje visokozmogljive aplikacije želimo imeti tekoče animacije in gladko interakcijo. To lahko v idealnih pogojih dosežemo s šestdesetimi sličicami na sekundo, kar pomeni, da ima aplikacija ob vsakem ponovnem izrisu na voljo samo 16,66 ms časa, da izvede potrebno kodo. Če potrebuje

za kakšno funkcijo več kot toliko, pride do manjšega števila sličic na sekundo, kar povzroči zatikanje pri animacijah in interakciji. Hitrost naše aplikacije lahko preverimo z jezičkom `Timeline` v načinu `Frames` v katerem imamo za vsak izris prikazane izvedene metode naše kode ter koliko časa je vsaka od teh metod potrebovala za izvedbo. Na podlagi teh informacij lahko hitro ugotovimo, kateri del naše kode upočasnjuje ponovne izrise.

Na čas izvedbe ponovnih izrisov pa ne vpliva samo naša koda JavaScript, temveč tudi oblikovanje CSS. Naprednejše metode oblikovanja lahko pri izrisu vzamejo velik delež procesorskega časa za svojo realizacijo. Primer takega oblikovanja so zaobljeni robovi v kombinaciji s senco. Za lažjo detekcijo oblikovnih problemov imamo v Chrome DevTools možnost omogočitve neprekinjenega izrisovanja aplikacije. V desnem zgornjem kotu okna se pojavi okvir, v katerem je prikazan graf časov trajanj zadnjih nekaj izrisov. Prednost tega je, da lahko poljubno skrivamo elemente in oblikovne sloge, pri čemer vidimo direktni odziv na spremembo v grafu. Tako hitro odkrijemo slabosti in izboljšave našega oblikovanja. Druga možnost v DevTools je sposobnost prikaza območja, ki je bilo ponovno izrisano. Večje ko je območje, več časa je bilo porabljenega za izris, zato je naša prioriteta da območja čim bolj zmanjšamo. Najbolj pogosto se ti problemi pojavijo pri pomikanju okna aplikacije.



Slika 4.5: Prikaz ponovno izrisanih elementov.

4.4 Sistem sporočil o napakah

Testiranje je eden ključnih postopkov pred vsako izdajo nove različice naše aplikacije, vendar nam vseh napak v tej fazi ne uspe vedno odkriti v tej fazi. Zato je priporočljivo, da ima naša aplikacija vgrajen samodejni način sporočanja napak na strežnik. Ko pride do napake pri izvajanju kode, se sproži izjema, ki jo lahko ujamemo z globalno metodo `onerror`. Pri tem dobimo podatek o vrstici, v kateri se je zgodila napaka, in sporočilo napake. Ne moremo pa izpisati sledi klicev. Če napako ujamemo v segmentu `try/catch` nam je ta možnost ponujena, vendar je koda zelo neberljiva, če mora vsaka funkcija vsebovati segment `try/catch`. Temu se lahko izognemo s posebno metodo, ki ob zagonu aplikacije vsako metodo objekta dinamično objame s segmentom `try/catch`. Tako je naša koda berljiva, prav tako pa nam ni treba spreminjati že obstoječe kode. V tej posebni metodi določimo še klic na strežnik, ki se kliče ob vsaki napaki in sporoči vse informacije o izjemi in njeno sled klicev. Kot vemo, je izvorna koda obdelana z orodjem Closure Compile, kar predstavlja dodatne probleme. Sporočilo o napaki vsebuje številko vrstice in stolpca v kodi, kjer je prišlo do izjeme. Če ti dve številki prevedemo na obdelano kodo, je skoraj nemogoče razbrati, kje se je napaka zgodila in zakaj. Zato si pomagamo z datoteko source map, ki je ustvarjena po obdelavi naše izvorne kode z orodjem Closure Compile. S pomočjo te datoteke lahko podatek o številki vrstice in stolpca preslikamo v izvorno neobdelano kodo, kar nam omogoča enako razhroščevanje kot v razvojnem okolju.

Poglavje 5

Sklepne ugotovitve

V okviru diplomske naloge smo razvili spletno aplikacijo, ki rešuje glavni problem sprememb na spletnem portalu YouTube. Namenjena je uporabnikom, ki intenzivno uporabljajo spletni portal YouTube in želijo biti ves čas na tekočem o novih posnetkih svojih naročnin. Aplikacijo smo razvili v tehnologijah HTML5, JavaScript in CSS3. S pomočjo treh različnih platform smo uporabnikom omogočili uporabo svoje aplikacije ob različnih priložnostih, glede na okoliščine, v katerih je uporabnik.

Razvoj aplikacije je potekal relativno hitro, saj sem že imel predznanje programiranja v programskih jezikih JavaScript in HTML5 in sem ga med samim razvojem nadgrajeval. Najprej se je celotna aplikacija razvila samo za eno platformo, potem pa so bile narejene ustrezne prilagoditve, da sta bili podprti še ostali dve platformi. Nekaj začasnih težav je bilo tudi s strežniškim delom aplikacije, predvsem pri strukturi podatkovne baze, v katero so se shranjevali videni posnetki uporabnikov. Z ustrezno prilagoditvijo strukture pa se je tudi ta problem rešil. Končni koraki razvoja so bili predvsem v poglobitvi v iskanje performančnih težav aplikacije ter odpravljanje teh problemov. To delo je naporno in zahteva veliko časa ter predvsem znanja s tega področja. Le-to je bilo pridobljeno z ogledom različnih video posnetkov Googlovih konferenc, ki so namenjene razvijalcem.

Velik poudarek razvoja je bil predvsem na uporabniški izkušnji z aplika-

cijo. Ta je morala biti tekoča in enostavna, pri tem pa boljša od tega, kar ponuja spletni portal YouTube. Menim, da smo to dosegli, saj je aplikacija po objavi na spletni trgovini Google Chrome Store dobila pozitiven odziv in zajetno količino uporabnikov. Večina le-teh je mnenja, da od kar so začeli uporabljati našo aplikacijo nimajo več potrebe po uporabi spletnega portala YouTube. To potrjuje tudi statistika. Aplikacijo trenutno tedensko uporablja 75.000 uporabnikov, skupno število uporabnikov pa je že 120.000. Zanimiv je tudi podatek, da v povprečju uporabnik aplikacijo uporablja 3 ure in 17 minut, kar potrjuje namenskost le-te.

Aplikacijo je mogoče v prihodnosti še dodatno razširiti. Glavna razširitev bi bila podpora aplikacije na mobilnih napravah z namensko aplikacijo. Trenutno je možno aplikacijo uporabljati s pomočjo vgrajenega brskalnika vendar to ne omogoča prave uporabniške izkušnje in izrabe vseh razpoložljivih virov mobilne naprave. Sledile bi razširitve aplikacije z dodatnimi pogledi video posnetkov in različnimi temami, saj nekateri uporabniki rajši uporabljajo svetlejšo temo kot temnejšo. Zaledje aplikacije bi bilo mogoče prenesti na Google App Engine, kar bi zmanjšalo mesečne stroške delovanja aplikacije, prav tako pa bi zmanjšalo izpade, do katerih lahko pride z uporabo kolokacije in odpovedjo strojne opreme. Prav tako bi lahko dodali podporo za naprave, ki omogočajo predvajanje video posnetkov na TV-zaslonih. Trenutno je podprt Chromecast, lahko pa bi dodali še podporo za Apple TV in Roku Box. V kombinaciji z mobilno aplikacijo se uporabnost naše aplikacije dvigne na višji nivo ter omogoči res enostavno in kakovostno spremljanje video posnetkov.

Literatura

[1] M. Pilgrim, *HTML5: Up and Running*, 2010, pogl. 3.

[2] T. Ericsson,"Front-end website performance optimisation", *Optimising the front-end performance of Swedbank's website*, 2013, str. 23–37.

[3] D. S. McFarland, *JavaScript & jQuery: The Missing Manual, Second Edition*, 2012, pogl. 4.

[4] M. Zandstra, *PHP Objects, Patterns, and Practice*, 2010, str. 361–378.

[5] T. Barker, *Pro JavaScript Performance: Monitoring and Visualization*, 2012, str. 109–137.

[6] (2012) Chrome Dev Tools: Networking and the Console. Dostopno na: http://buzzyr.com/post/38053166272/chrome-dev-tools-networking-and-the-console

[7] (2014) Programming languages used in most popular websites. Dostopno na: http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

[8] (2014) jQuery Usage Statistics. Dostopno na: http://trends.builtwith.com/javascript/jQuery

[9] (2014) Chrome Apps - Disabled Web Features. Dostopno na: https://developer.chrome.com/apps/app_deprecated.html

[10] (2014) OAuth 2.0. Dostopno na:
https://developers.google.com/accounts/docs/OAuth2

[11] (2014) Google Trends. Dostopno na:
http://www.google.com/trends/explore#cmpt=q

[12] (2014) Seznam najpopularnejših spletnih strani. Dostopno na:
http://www.alexa.com/siteinfo/youtube.com

[13] (2014) jQuery. Dostopno na:
http://jquery.com/

[14] (2014) Chrome Legacy Packaged App. Dostopno na:
http://developer.chrome.com/extensions/apps.html

[15] (2014) Chrome Packaged App. Dostopno na:
http://developer.chrome.com/apps/about_apps.html

[16] (2014) TweetDeck. Dostopno na:
https://tweetdeck.twitter.com/

[17] (2014) NetBeans. Dostopno na:
http://www.netbeans.org/

[18] (2014) Adobe Photoshop. Dostopno na:
http://www.adobe.com/si/products/photoshop.html

[19] (2014) TortoiseSVN. Dostopno na:
http://tortoisesvn.net/

[20] (2014) Google Chrome. Dostopno na:
http://www.google.com/intl/sl/chrome/

[21] (2014) Acid1, Acid2, Acid3. Dostopno na:
http://www.acidtests.org/

[22] (2014) Test 262. Dostopno na:
http://test262.ecmascript.org/

[23] (2014) Google Chrome statistika. Dostopno na:
http://en.wikipedia.org/wiki/Google_Chrome

[24] (2014) SPDY. Dostopno na:
http://www.chromium.org/spdy/spdy-whitepaper

[25] (2014) Chrome Web Store. Dostopno na:
https://chrome.google.com/webstore

[26] (2014) Chromecast. Dostopno na:
https://www.google.com/intl/sl/chrome/devices/chromecast/

[27] (2014) YouTube Data Api v2. Dostopno na:
https://developers.google.com/youtube/2.0/developers_guide_protocol

[28] (2014) YouTube Data Api v3. Dostopno na:
https://developers.google.com/youtube/v3/

[29] (2014) Closure Compiler. Dostopno na:
https://developers.google.com/closure/compiler/

[30] (2014) JSMin. Dostopno na:
http://www.crockford.com/javascript/jsmin.html

[31] (2014) YUI Compressor. Dostopno na:
http://yui.github.io/yuicompressor/