Tallinna Ülikool

Informaatika Instituut

Veebiteenuse arendamise teekaart Rada7.ee näitel

Bakalaureusetöö

Autor: Kirill Milovidov

Juhendaja: Jaagup Kippar

Autor: ..... „ ..... “ 2015
Juhendaja: ..... „ ..... “ 2015
Instituudi direktor: ..... „ ..... “ 2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Kirill Milovidov (sünnikuupäev: 1. august 1987)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebiteenuse arendamise teekaart Rada7.ee näitel” mille juhendaja on Jaagup Kippar, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, _____

allkiri ja kuupäev

Sisukord

Sissejuhatus

Enamus tänapäeval loodud teenusepõhiseid tooteid töötavad interneti vahendusel, läbi mingi kasutajaliidese. Sealjuures on kasutajaliides selles kontekstis üsna abstraktne mõiste ja ei pea olema ainult arvuti, telefon või televiisor. Kasutajaliides võib olla ka kohvimasin, mis näiteks keedab teatud viisil kohvi, vastavalt kasutaja digitaalses Google kalendris olevatele sündmustele. Palju igavaid kohtumisi: kange kohv, meeldiv hommikusöök vana sõbraga: kohv piimaga. Kirjeldatud stsenaariumis toimub infovahetus mitme erineva instantsi vahel: kasutaja suhtleb algse rakendusega, mis omakorda küsib Google-lt kasutaja kalendri kirjeid, kohvimasin suhtleb samuti algse teenusega, et teada saada vahepeal mingi algoritmi alusel kokku arvutatud hommikuse kohvi retsepti.

Kohvimasinat huvitab ainult, mis tulemuse äriloogika välja arvutas. Teda ei huvita muutujad, mis tulemuse arvutamiseks vajalikud on. Selge on see, et äriloogikat ei saa kohivmasinasse lisada, vaid peab olema mingi keskne teenus, mida kasutajaliidesed saavad pruukida. Äriloogika peaks asuma tsentraalses kohas ning sellega suhtlemine tuleks teha võimalikult lihtsaks.

Teenusepõhine lähenemine ei ole iseenesest midagi uut, kuid aktuaalne lahendus tänapäeval üha süvenevas kasutajaliideste rohkuses. See võimaldab ühendada erinevad seadmed ja kasutajad ühe platvormi alla. Igale kasutajaliidesele (rakendusele) ei pea kaasa arendama tervet äriloogikat, palju lihtsam on hoida töös keskne teenus, mida saavad erinevad kliendid enda liidese vajadustele vastavalt kasutada. Eelpool toodud näite puhul saaks telefonikliendis toimingutest ülevaade aga kohvimasinas saaks keeta ainult kohvi.

Kuid selline masinvaheline suhtlus seab tingimusi, mida tuleb kindlasti täita, et teenusega ühenduvad teised teenused või kliendid saaks alati standardiseeritud vastused ja seisu.

Käesoleva bakalaureuse töö eesmärgiks on luua teekaart portaali Rada7.ee veebiteenuse esmase arendamise etapi jaoks. Selleks analüüsitakse erinevaid raamistikke tehnilise lahenduse teostamiseks, veebiteenuse standardeid, mida teenus rakendama peaks ja pakutakse omapoolne nägemus milline nendest võiks olla Rada7.ee jaoks parim lähenemine. Standardite uurimisel käsitletakse olemasolevaid suuremaid platvorme, et leida levinumad ühisosad,

millest saaks eeskuju võtta oma teenuse väljatöötamisel. Täpsemaks ülevaateks luuakse ka lihtne töötav prototüüp veebiteenusest ja seda kasutavast kliendist. Prototüübis käsitletakse veebiteenuse üldisemaid ja tähtsamaid osasid, tulenevalt Rada7.ee vajadustest.

Rada7.ee veebiarenduse kohta on eelnevalt loodud üks bakalaureusetöö (Nurme 2006). Erinevalt teisest lõputööst, kus kirjeldati eelnevalt loodut, on käesoleva bakalaureusetöö eesmärk ette planeerida ja ära nimetada vajalikke tehnilisi aspekte ja samme, mida on vaja selleks ette võtta, et valmistada veebiteenus.

1. Mis on Rada7.ee?

Rada7.ee on Eesti suurim alternatiivmuusika portaal, mis tegutseb juba aastast 1999. Esialgu ainult staatilise lehena hot.ee serveris, mida uuendati FTP kaudu uusi versioone üles laadides, ning veidi hiljem interaktiivse portaalina, kus lisaks artiklitele saavad kasutajad ise vestlustes osaleda ja arutelusid luua. Hiljem lisandusid ürituste kalender ja (eelkõige muusikariistade) turg ning bändikaaslaste otsimine. Juba algusest on Rada7.ee keskmes olnud sisutootmine, mille ümber tekib huvitav arutelu. Portaalis avaldatakse regulaarselt intervjuusid, ülevaateid, tutvustusi, arvustusi ning värsket muusikat. Paljud artiklid on kirjutatud ajakirjanduses kui ka kirjanduses tuntud autorite poolt. Rada7.ee kasutajate hulgas on palju kohaliku muusikaelu võtmeinimesi.

Tänapäeval keskendub Rada7.ee endiselt eelkõige huvitavate artiklite tootmise ning avaldamisega. 2015. aastal pärjati Rada7.ee „Aasta meediakanal” tiitliga Eesti Muusikaettevõtluse Auhindade tseremoonial, edestades erakapitalil põhinevat veebiportaali Delfi kui ka avalik-õiguslikku Eesti Televisiooni (kultuuri osa), kes olid samas kategoorias teised nominendid. Rada7.ee eestvedajate näol on tegemist vabatahtliku võistkonnaga, kes teevad seda oma päevatöö kõrvalt.

2. Rada7.ee ärikiht

Rada7.ee ärikiht koosneb tänaseni paljuski sellest, mida kirjeldati Nurme 2006 bakalaureusetöös. Paljud selles töös väljatoodud tulevikuideed on rakendatud, mõned edukalt ja mõned vähem edukalt.

Samas on Rada7.ee arendamise protsesside käigus ilmnenud, et parim viis ärikihilisi muudatusi teha on väikeste sammudena, mitte korraga suure tükina. Sedasi on võimalik ära testida millised muudatused on vajalikud ja toimivad, ning millised on poolikud või ebavajalikud. Lisaks on sellise graafikuga tihedamini võimalik saavutada väikeseid võite.

Väga palju uusi ärinõudeid käesolevas töös ei kirjeldata, vaid keskendutakse tehnilisitele lahendustele. Uue versiooni eesmärk on viia kogu vana funktsionaalsus uuele tehnilisele baasile (ingl *technology stack* või lihtsalt *stack*) ja veebtieenuse struktuurile.

2.1. Koostisosad

Rada7.ee koosneb paljudest osadest, mis on omavahel läbipõimitud.

2.1.1 Kasutajate süsteem

Portaali selgrooks on kasutajate süsteem, kus igal kasutajal saavad olla erinevad ligipääsuõigused. Nendeks on:

- „superman” – kõik administraatori õigused
- kasutajate info muutmine, blokeerimine, aktiveerimine
- reklaamide lisamine ja muutmine
- artiklite ja uudiste muutmine ja avaldamine
- artistide haldamine

Samuti on kasutajatel erinevad olekud:

- aktiveerimata
- aktiveeritud
- blokeeritud

Kasutajad saavad üksteisele saata sõnumeid. Sõnumid kuvatakse teemapõhiselt, ühes teemas

on korraga n arv sõnumeid.

Kasutaja profiil saab olla ka seotud autori objektiga, mida saab ühendada artikli objektiga.

Kasutajad saavad sisse logida sisestades oma kasutajanime ja salasõna kombinatsiooni, või kasutada „jäta mind meelde” funktsionaalsust, mille puhul salvestatakse kasutaja arvutisse *cookie*, mille olemasolul ja kehtimisel logitakse kasutaja iga uue sessiooni alguses automaatselt sisse.

2.1.2 Foorum ja teema objektid

Teine oluline aspekt on foorumite süsteem, mille najale on ehitatud teemad, artiklid, üritused, artistid, loosimised ja uudised. Foorumid on oma olemuselt kategooriad, mis määravad vestluse suuna ja kus asuvad teemad. Foorumites olevad teemad on sorteeritud viimati tehtud postituse aja järgi. Igale foorumile saab määrata kasutajate seast moderaatoreid, kes saavad teemasid ja postitusi kustutada, muuta, sulgeda/avada, liigutada. Artiklite ja uudiste avaldamise või muutmise õigus sellistel kasutajatel pole, isegi kui artikkel asub modereeritavas alamfoorumis.

Teema objekt kujutab endast mingit kasutajapoolset teemapüstitust või küsimust, nagu klassikalises foorumis tavaks. Seejärel saavad teised kasutajad samasse teemasse postitada.

Artikkel, üritus või artist on kõigest „teema” objekti alamobjekt. Iga selline alamobjekt on alati seotud mingi teema identifikaatoriga ning asub eraldi andmebaasitabelis.

Lisaks tavalistele postitustele saavad vastava õigusega kasutajad lisada teemadesse ka uudiseid või teemat postitades valida, et tegemist on uudisega. Viimase variandi puhul ei ole tegemist teema objekti alamobjektiga, nagu seda on artikkel, üritus ja artist. Kuna uudis saab olla ka tavaline postitus, siis uudise andmeobjekt saab korraga olla seotud teema või postituse objektiga. Uudiseid kogutakse eraldi rubriiki ning esilehele.

Vastava õigusega kasutajad saavad teemade külge ühendada loosimisi. Neid saab lisada läbi eraldi lehekülje, määrata saab alguse ja lõpu aja. Süsteem avaldab õigel ajal loosimise, mis ilmub teemasse koos teiste postitustega kronoloogiliselt. Seni kuni loosimine on avatud, saavad kõik soovijad end loosimisele registreerida läbi loosi postituses oleva vormi. Kui loos on läbi saanud, suletakse selle registreerimine. Administraator peab seejärel laskma süsteemil

välja valida võitjad ning siis nende nimed avaldama samasse teemasse kuhu lisati loosimine.

Teema üks alamobjekte on artikkel. Artikli puhul paigutatakse kõigepealt teema objekt andmebaasi tabelisse ja siis luuakse artikli tabelisse eraldi sissekanne, kus asuvad spetsiifiliselt artikli jaoks vajalikud väljad, milleks on avaldamise kuupäev ja sissejuhatus. Nagu eelpool mainitud saab siduda autoriobjekte artiklitega, mis tähendab, et artikli kuvamisel kasutatakse autori nime, tema kaustajanime asemel. Artiklite nimekirja saavad kasutajad sirvida eraldi aadressilt ning viimased artiklid on kuvatud ka esilehel.

Artiklitele on võimalik lisada oma alamobjekt, erileht. Seda objekti eristab tavalisest artiklist võimalus anda talle unikaalne URL. Erilehti kasutatakse näiteks muusikaplaatide eelkuulamiste korraldamiseks. See kujutab endast võimalust kuulata piiratud aja jooksul mingit plaati läbi Rada7.ee lehe. Erilehe lisamisel saab määrata selle avaldamise ja lõppemise aja. Taustasüsteem suudab õigel ajal eelkuulamise avaldada ja sulgeda, avaldamisel lisatakse iga lehekülje päisesse pilt, millele vajutamisel suunatakse kasutaja eelkuulamise erilehele.

Ürituse ja artisti objekt toimib täpselt samamoodi nagu artikli objekt. Ürituse lisamisel saab määrata asukoha, kuupäeva kui ka muud lisainformatsiooni. Lisades esinejate nimekirja nimesid, mis esinevad Rada7.ee artistide andmebaasis, seotakse vastavad objektid automaatselt ürituse külge. Kasutajad saavad märkida, et nad on seda üritust külastanud või plaanivad külastada.

Artistiprofiilis on võimalik määrata artisti nimi, *slug*, lisainfo, sotsiaalmeedia lingid, koduleht ja SoundCloud-i teenuse aadress. Viimase olemasolu puhul käib Rada7.ee süsteem aeg-ajalt SoundCloud-ist andmeid pärimas, et koostada nimekiri andmebaasis olevate artistide lugudest. Artistide jaoks mõeldud eraldi koondlehel asub nimekiri, kus kuvatakse „aktiivsed” artistid. Igal artistil on kolm erinevate andmete uuendamise viimase kuupäeva välja, mille järgi neid filtreeritakse aktiivseteks ja väheaktiivseteks.

2.1.3 Kalender

Kalender koondab endasse ürituste nimekirju mida saab filtreerida konkreetsete kuupäevade ja asukohtade järgi. See on üks Rada7.ee populaarsemaid rubriike.

2.1.4 Sildid

Sildisüsteemi kaudu on võimalik lisada teemaobjektidele erinevaid silte ehk märksõnu, mille kaudu saab objekte katalogiseerida. Igal sildil on 2 tähtsat andmepunkti: pealkiri ja *slug*. Viimane on pealkirjast moodustatud inimloetav identifikaator. *Slug* moodustatakse asendades kõik mitte-ladina tähed ladina tähtedega. Näiteks eesti keele tähestikus on selleks ööüõ, mis asendatakse oauo-ga. Kõik lisamärgid (nt ', &, /, tühik jne) eemaldatakse või asendatakse märgiga „-”. Objektid, mis on märgitud siltidega „löök” ja „look”, on leitavad sama päringu järgi, kuna mõlemal on *slug*-iks „look”.

Kuigi algselt sai iga kasutaja ise silte lisada, siis tänapäeval see toimib ainult taustal: süsteem leiab ise märksõnu ja lisab neid vastavalt vajadusele. Praegu toimib automaatne sildistamine artistide ja ürituse lisamisel, kus sildiks tehakse artisti nimi, ja üritusele lisatud asukoht. Seda viimast kasutatakse eraldi „klubi” vaatena, mille alla koondub kõik selle sildiga seotud üritused. Näiteks /klubi/von-krahl aadressilt leiab üritused, millele on lisatud asukohaks (ja süsteemi poolt sildiks) „Von Krahl”.

2.1.5 Otsing

Rada7.ee otsimootori kaudu saab otsida teemasid ja kasutajaid. Otsimisel on võimalik kasutada järgnevaid filtreid:

- otsisõna
- kasutajanimi
- ajavahemik
- alamfoorum
- teema tüüp (teema, artist, artikkel, üritus)

Otsisõna puhul kasutatakse *full text* otsingu tehnikaid, mida pakub andmebaasimootor.

2.1.6 Reklaamid

Administraatorid saavad lisada reklaamibännereid, millele kogutakse kuvamiste ja vajutamise statistikat. Reklaame kuvatakse igal leheküljel kuni viis tükki, igal reklaamil on oma „kaal” mille järgi on „raskema” kaaluga reklaamidel lihtsam lõplikku kuvamise nimekirja saada.

Reklaami lisamisel saab ka selle kuvamise statsionaarseks muuta, mille puhul kuvatakse bännerit igal lehelaadimisel.

2.1.7 Taustasüsteemid

Rada7.ee hoiab töös paljud tausta/abisüsteemid, mille olemasolu pole tavakasutajale ilmne. Näiteks kasutajate sisendi puhastamise ja töötlemise eest vastutab nii raamistiku poolt pakutavad teegid kui ka autori loodud BB koodi *parser.* BB kood on foorumites kasutatav HTMLi alamkeel, mille kaudu saab postitustele lisada erinevaid tekstistiilide vorminguid (kursiiv, allajoonitud, jäme), pilte ja videosid. Osad nendest võimalustest on piiratud ainult administraatoritele.

Selleks, et lehelaadimine toimuks kiiremini, on paljud andmebaasipäringute tulemused salvestatud vahemällu. Praegune lahendus väga palju kiirust juurde ei anna, sest vahemäluna kasutatakse failisüsteemi, mis on üks aeglasemaid lahendusi selles vallas. Paraku on vajadus selle järele suure kasutajatehulga tõttu. Kuna Rada7.ee paikneb teistega jagatud virtuaalserveris, on andmebaasi päringute arv sekundis piiratud. Just andmebaasi koormuse vähendamiseks ongi tõstetud alamfoorumite ja teemade sisu vahemällu. Selline lahendus lahendab andmebaasile ligisaamise probleemi.

JavaScripti kokkupanemiseks ja haldamiseks on kasutatud autori poolt loodud JSMan nimelist teeki, mida on põhjalikult tutvustatud autori poolt sellele eraldi loodud seminaritöös (Milovidov, 2015).

Osad pikemalt töötlust protsessid on paigutatud töötama taustal, neid käivitab eraldi taustatööde süsteem. Näiteks on pikemalt töötlemist vajav protsess SoundCloudist artistide lugude nimekirja allalaadimine.

2.2. Miks mitte jätkata praeguse tehnilise lahendusega?

Praegune versioon on üles ehitatud CodeIgniter raamistikule. Tegemist on üsna populaarse raamistikuga, mis tänapäeval on paraku tehniliselt väga vananenud. See on hea raamistik neile, kes alles alustavad veebiraamistikke kasutama. Baaskomplektina pakub ta selliseid tööriistu:

- klasside automaatne laadimine (klassid peavad asuma teatud kataloogides ja nimeatud

teatud viisi)

- andmebaasi (mudeli) kiht (eelkõige MySQL toega)
- algeline vaadete (ingl *template/view*) tugi
- *controller* ehk objektid, mis vastavad mingile konkreetsele URL-ile
- hulk väikesed abistavaid funktsioone failide lugemiseks, inglise keele käänamiseks, sisendi ja väljundi filtreerimiseks (URL/e-mail) jne

Viimased 3 aastat on raamistiku arendamine sisuliselt seisnud, kuna omanikfirma Ellislab otsis, kellele CodeIgniter-i projekt üle anda. Peale pikki otsinguid valiti selleks British Columbia Institute of Technology. Raamistik ei toeta tänapäeval PHP-s laialt kasutusele võetud PSR standardeid (PHP Framework Interop Group, 2015).

Kuna raamistik eeldab väga konkreetset endapoolset (faili)struktuuri, on väga lihtne suure projekti puhul koodi dubleerida, sest tehniliselt pole võimalik erinevaid objekte väga lihtsalt üksteisega siduda. Igasse klassi, mis ei ole *model* ega *controller* tüüpi, tuleb lisada otsene sõltuvus CodeIgniter-i instantsile, mille tõttu on vaja adapterklasse (näide autori JSMan-i seminaritöös Smarty jaoks) või siis lisada instants esialgsesse klassi. Raamistik ei arvesta eriti muude taustal olevate kihtidega, mis ei ole *controller* ega *model*. Kuid need on kõigest kaks kihti, mis on vajalikud ainult mingile URLile vastuse genereerimise jaoks. Enamus ärikihi koodist ei peaks asuma ainult nendes kahes tüübis.

Kõik see on viinud on selleni, et Rada7.ee projektis on väga palju ebavajalikku ja üleliigset koodi, mille üle puudub ülevaade. Samas raamistiku paindumatuse tõttu ei olegi paljud halvad lahendused teisiti teostatavad.

Selline seis ei võimalda ka uusi inimesi projekti juurde tuua, kuna puudub selge ja tuttav arusaam kuidas ja mis standardite järgi koodi ning uusi mooduleid kirjutada. Paljud komponendid (nt kasutajate haldus, vahemälu süsteem) ei ole uutele tulijatele tuttavad ja põhjuseta keerulised.

2.3. Tuleviku vajadused

Uus tehnoloogiline alus peab pakkuma vabadust erinevate võimaluste teostamisel. Ideed ei

tohiks jääda tehnoloogia võimetuse tõttu teostamata.

Kood peaks olema kirjutatud teatud reeglite järgi. Tähtis on, et kogu süsteem järgiks *convention over configuration* printsiipi. See tähendab, et arendajate kasutada on väljakujunenud viisid teatud probleemide lahendamiseks. Näiteks Javas on välja mõeldud ja kinnitatud standardid JSR (*Java Specification Requests*, meenutavad olemuselt ISO standardeid), mis selle printsiibi täitmist lihtsustavad. See soodustaks ka uutel arendajatel, juhul kui neid lisandub, kiiremini sisse elada.

Väga lihtne on planeerimise ajal selgelt mõelda kuidas koodi kirjutada, kuid järelhoolduse ja uute moodulite arendamisel on see kõik loomulikult ununenud. Siin aitab *convention over configuration* printsiip, mis seab sisse väga konkreetsed reeglid kuidas midagi tegema peaks ning kõrvalkalde puhul on tajutav, et selline tehniline lahendus ei sobi.

2.4. Uued ärikihilised ideed

Täiesti uutest äripoolsetest ideedest oleks võimalik rakendada muusikapleier, mida kasutajad saavad kasutada samal ajal kui nad portaalis ringi liiguvad, ilma, et muusikasse tekiks pause ja lehekülge peaks jaotama HTML raamide sisse.

Kuna kõik andmed on saadaval läbi REST veebiteenuse, on võimalik andmeid jagada ka muudesse rakendustesse, kui www.rada7.ee. Olgu selleks Rada7.ee poolt loodud rakendused (näiteks Android/iOS platvormidele loodud rakendus) või mõne kolmanda osapoole lahendus. Algne infoallikas oleks alati sama ja keskne, seega oleks veebiteenust tarbivatel rakendustel ainult vaja luua enda olemusele spetsiifiline liides (ingl *interface*).

Rutates veidi ette, võib öelda, et valikus olnud raamistikud toetavad WebSocket protokolli, mille kaudu on võimalik TCP ühenduse tüübi järgi kliendi ja serveri vahel suhelda, ilma, et selleks peaks eraldi HTTP päringuid looma (Internet Engineering Task Force, 2011). Seda kasutades on võimalik saata näiteks uuendusi rakendustesse, mis on seadistatud selliseid sõnumeid saama. Näiteks eraldi rakendus nutiseadme platvormil, mis saab uuendusi kui Rada7.ee-s ilmub uus artikkel või kasutaja on saanud uue sõnumi.

3. Veebiteenused ja kliendid

Veebiteenus kujutab endast serverit, mis tagastab päringutele masinloetavat infot, nt JSON, XML, yaml jne formaadis. Laiemalt tähendab see kahe masina vahelist suhtlust üle võrgu (The World Wide Web Consortium, 2014). Käesolevas töös kasutatakse veebiteenuse REST (*representational state transfer*) standardit.

3.1. REST-i tutvustus

REST teenus töötab läbi HTTP (*Hypertext Transfer Protocol*) protokolli ja on olekuta. Oleku puudumine tähendab, et kasutaja sessiooni ei ole olemas (The World Wide Web Consortium, 2004). Autentimismehanismid peavad tagama, et kasutajal on ligipääs ressursidele, kuid süsteem ei tohiks selle põhjal tekitada sessiooni. Akronüümi viimased kaks tähte ST viitavad sõnadele *state transfer* ehk oleku ületoomisele: kliendis nähtav seis tekitatakse serverist tulnud vastustest ning ka klient ise võib saata oma oleku REST teenusele. Selline arhitektuur kulutab igal päringul ressurssi, et veenduda kasutaja ligipääsuõigustes. (Fielding, 2000, 5.1.3) Sellest piirangust möödapääsemiseks kasutatakse tihti mitut autentimissüsteemi korraga, kus keerukus on neid süsteeme isoleerida, et nad üksteist ei takistaks. Lahendusena kasutatakse ka sessiooni loomist, mis rikub oleku puudumise nõuet.

REST erineb teistest veebiteenuse tüüpidest selle poolest, et ta pakub nimeliste *endpoint*-ide kaudu kasutajale ligipääsu rakenduses olevatele ressursidele, mitte rakenduse äriloogika programmiliidesele ehk *application program interface* ehk API (The World Wide Web Consortium, 2004). Viimasel juhul oleks tegemist SOAP (*Simple Object Access Protocol*) teenusega. *Endpoint* kujutab endast mingit HTTP kaudu ligipääsetavat aadressi, mis vastab ressursiga.

HTTP spetsifikatsioon määrab verbid, mida saab päringutes kasutada. Sellise semantika kaudu on võimalik samalt *endpoint*-ilt erinevat infot saata ja saada. Näiteks GET verbi puhul saadakse küsitava ressursi baasinfo, PUT puhul saadetakse kliendi poolt muudatusi veebiteenusele, DELETE puhul tehakse kustutamise päring (Fielding & Reschke, 2014). Sedasi on võimalik hoida *endpoint*-ide arv võimalikult väike ja semantiliselt arusaadavamana (näide 1).

GET	/articles/6	Pärib artiklitest ühe tulemuse, mille identifikaator on 6
PUT	/articles/6	Saadab muudatusi artikli ressursi uuendamiseks
DELETE	/articles/6	Kustutab märgitud identifikaatoriga artikli

Näide 1. *Päringud samale aadressile erinevate verbidega, erinevate tulemustega.*

Sellele lisaks peab kindlasti kasutama HTTP päringute tulemuste koode (*HTTP status code*). Need kolmekohalised numbrid sisaldavad endas päringu tulemuse tähendust. Koode jagatakse esimese numbri alusel 5 erinevasse gruppi (Fielding & Reschke, 2014):

- 1xx – päring toimus, jätkatakse töötlusega
- 2xx – päring lõppes edukalt ja süsteem on sellest aru saanud ning vastu võtnud
- 3xx – suunamine
- 4xx – kliendipoolne viga: vale sisend, tulemust ei ole olemas või pole leitav
- 5xx – serveri viga, teenuste töös esines häire, kuid kasutaja sisend on korrektne

Iga REST päring peab tagastama ühe sellise koodi, mis on kooskõlas tulemuse semantikaga. Üks suurimaid vigu, mida tihti tehakse, on koodi „200 OK” saatmine kuigi tegelikult päringus ilmnes mingi viga. Selline väljund ei ole masinloetav ja tekitab tõrkeid kliendipoolses süsteemi toimimises.

3.2. Andmeformaadid

REST teenus võib tagastada vastuseid erinevas andmeformaadis, levinumad nendest on JSON (JavaScript Object Notation) ja XML (Extensible Markup Language). JSON on loodud Javascripti objektina, kus oluliseks on lihtne loetavus inimese kui ka masina poolt (ECMA International, 2015).

XML on range semantilise ülesehitusega dokument, kus sisuks on märgendid. XML-i teeb võimalusterohkuseks erinevate valideerimiskriteeriumide rakendamine (Kippar, 2009).

3.3. Klient ja *Single Page Application*

Kuna veebiteenuse päringute tulemused on eelkõige masinloetavad, siis inimese jaoks mugavamaks lugemiseks on vaja mingi kliendi olemasu, mis koondab päringute tulemused

sellele kliendi kasutajaliidese jaoks vajalikku formaati. Just see ongi *Single Page Application* (edaspidi SPA) ülesanne.

Klassikalises veebirakenduses genereerib server kasutaja sisestatud aadressipäringu peale tulemuse, milleks on HTML. Võib öelda, et sellises rakenduses on olekuks veebiaadress (URL). SPA kasutajaliides kujutab endast samuti veebilehte, ta erineb ainult oma tehnilise põhja poolest. Kui veebilehitseja suunatakse mingile SPA kliendi aadressile, püüab sellel domeenil töötav SPA suunamismehanism (*router*) selle kinni ja kuvab sellele aadressile vastava oleku, mis on omakorda kogum veebiteenuse päringutulemus(t)est ja JavaScripti mudelitest. Need päringutulemused ja mudelid pannakse seejärel kokku, et tekitada HTML väljund, mida veebilehitseja oskab välja kuvada.

See tähendab, et rakenduse olekute vahel liikumine saab toimuda ilma kogu lehte uuendamata, uuendades ainult teatud blokke. Sedasi on võimalik pakkuda sujuvamat kasutajakogemust. Hästi teostatud JavaScripti rakendust on võimalik ka üle viia (*port*ida) nutiseadme platvormile, suurtemate lisakuludeta.

Rada7.ee SPA oleks sellisel juhul kättesaadav praeguselt portaali aadressilt, www.rada7.ee.

4. Raamistike valimine

Raamistiku valimine on edasise arendamise alustala. Raamistik peaks täitma peatükis „Tuleviku vajadused” nimetatud eesmärke. Valiku kriteeriumid on osaliselt subjektiivsed, sest lähtuvad autori olemasolevast teadmistepagasist. Kõik loetletud raamistikud, ja neid mahutav keskkond, töötavad Windows, Linux ja Mac OSX operatsioonisüsteemides.

4.1. Valiku kriteeriumid

Objektiivsete valikute kriteeriumideks on kasutusel Symfony raamistiku loojate poolt väljatöötatud nõuded. Nendest kõige olulisemad punktid oleks:

- raamistiku keele kogukond ja populaarsus
- jätkusuutlikus, tehniline võimekus ja filosoofia
- dokumentatsioon ja tehniline tugi
- turvalisus

Raamistik peab olema piisavalt populaarne ning selle ümber peab olema tugev kogukond, kes on selle edasiviivaks jõuks. Probleemide tekkimisel on lihtne abi leida, kuna sellisele küsimusele on juba kunagi vastatud. Või kui ei ole, on olemas piisavalt inimesi, kellelt sellise mure korral küsida on võimalik. Kogukonna suuruse hindamiseks on heaks teeviidaks otsing suurte arendusteemaliste portaalide andmebaasist. Kirjutamise hetkel on suurim neist „Stack Overflow” (Alexa, 2015). Kindlasti tasub uurida ka ametlikult kodulehelt kui tihti on erinevaid versioone ja uudiseid avaldatud, ning kui palju on aega möödas viimasest sellisest sissekandest. Kuigi see pole alati objektiine võib see anda päris hea ülevaate.

Jätkusuutlikkus ja tehniline võimekus on olulised punktid, kuna nad määravad järelhoolduse ning juurdearendamise lihtsuse. Kui uurimise käigus selgub, et tehnoloogiliselt on ideede rakendamine selle raamistikuga keeruline või võimatu, on alust arvata, et see pärsib ka edaspidist arengut (Symfony, 2015).

Kindlasti on väga oluline arendamise lihtsus ja väliste tööriistade olemasolu (selle alla kuulub osaliselt raamistiku ja programmeerimiskeele kogukond, kuna nemad on tööriistade tootjad).

Kõige tähtsamaks tööriistaks on arenduskeskkond, mida tasuks võtta ühe suure kogumina: selle all tasub mõelda mootorit, kus raamistik jooksma peaks; programmeerimiskeelt; dokumentatsiooni ja programmi, kus koodi kirjutatakse (*Integrated Development Environment* ehk IDE). Keskkonnas olevad muutujate ja lisade arv tasub hoida võimalikult minimaalsena. See ei tähenda, et keskkond peaks olema äärmiselt labane, pigem just nii võimas, et saab pakkuda lihtsaid asju, lihtsalt. Keskkonna ülesseadmine ei tohiks olla raske ega rusuv töö, mida ei taha keegi ette võtta.

Nagu eelnevalt mainitud, peaks raamistik seadma enda poolt väga selged nõuded ja standardid, kuidas erinevaid lahendusi luua. Ehk toetama *convention over configuration* lähenemist.

Subjektiivsetest valikutest on oluline arendajate eelnevad teadmised ja isiklikud eelistused valikus olevate raamistike kasutamisel.

4.2. Väljavalitud raamistikud

Kasutades eelnevas peatükis väljatoodud põhimõtteid, tasub valida nende raamistike vahel:

- Laravel 5 (PHP)
- Symfony 2.7 (PHP)
- Node.js baasil platvorm (JavaScript)
- Spring Framework 4.2 + muud Springi komponendid (Java)

Nimekirjas on bakalaureusetöö kirjutamise ajal saadaval olevad viimased versioonid või nende beetaversioonid.

4.2.1 Laravel ja Symfony

Nende kahe valiku suurim eelis on see, et nende kasutamine ei tooks kaasa praeguse serverikeskkonna radikaalsemat muutmist nagu seda nõuavad teised nimekirjas. Need raamistikud töötavad Apache HTTP serveris, nagu ka praegune Rada7.ee versioon. Selles serverikeskkonnas ei saa kasutada WebSocket protokolli, kuid see ei ole ka väga oluline punkt.

Muudes punktides vastavad need kaks raamistikku ideaalselt loetletud nõuetele. Symfony on tootnud väga suure koguse klasse, mida paljud, sh. Laravel, kasutavad oma paketis ja on seeläbi üks suuremaid PHP raamistike ja standardite suunamäärajaid.

Mõlemad toetavad *dependency injection* koodimustrit ja annotatsioone. Halvaks võib lugeda ebameeldivat keskkonna paigaldamist ja töötamise olemust. PHP toimib *script*-ina, URL-i päringu peale käivitatakse vastavad klassid. Klasside laadimine toimub spetsiifilise faili asukoha ettekirjutamisega või siis Composer tööriista kaudu, millega on võimalik PSR-0 ja PSR-4 standardiga klasse automaatselt laadida. Kui klasside struktuuri muudetakse (lisatakse, kustutatakse) siis tuleb käsurea kaudu genereerida automaatse laadimise nimekiri Composer-i kaudu uuesti. Võimalik kasutada ka annotatsioone/aspekte.

PHP töötab Apache HTTP serveris (võimalikud ka muud serverid), mis tähendab, et arenduskeskkonna ülesseadmisel on vajalik tekitada võimalikult sarnane produktsiooni keskkond nii PHP kui ka serveri konfiguratsioonis. See on väga tüütu ja keeruline hoida sünkroonis. Versioonide uuendamine on sellises keskkonnas ebamugav, kuna peab üle käima kõik konfiguratsioonifailid. Hea tööriist selle keerukuse lahendamiseks on virtuaalmasin, mis kasutab samasugust konfiguratsiooni nagu server, kus rakendus lõpuks töötama peab. Sellist võimalust pakub näiteks Vagrant (HashiCorp, 2015). 2015 aasta Stack Overflow uuringust selgub, et LAMP keskkond (akronüüm esimestest tähtedest: Linux, Apache HTTP, MySQL, PHP; pakett, mis võimaldab PHP-d kasutada) on üks ebameeldivamaid tööriistu. Vastanute seast 62.2% selle kasutajatest on väljendanud arvamust, et nad ei sooviks seda võimaluse korral edaspidi kasutada (Stack Overflow, 2015).

PHP koodikirjutamise keskkonnad (IDE) on üsna võimekaid, kuid kuna PHP oma olemuselt ei ole täiesti objekt-orienteeritud, siis paljude objektide kokkuviimine tuleb teha erinevate annotatsioonega koodis (näide 2). See ei ole küll kohustuslik, kuid kindlasti hea viis kuidas arendamise ajal selgemat ülevaadet saada.

```
/**
 * @param int $id
 * @return SoundCloudTrack[]
 */
public function getArtistTracksFromSoundCloud($id)
{
 // ...
        foreach ($data as $track) {
                $results[] = new SoundCloudTrack($data);
        }
        return $results;
}
```

***Näide 2**. Koodi meta-annotatsioonid enne meetodi definitsiooni. Sedasi teab IDE, mis tüüpi klassid on arrays ja oskab selle meetodi kasutamisel vihjata arrays oleva klassi meetodeid ja atribuute. Eeldab, et array-sse on lisatud ainult sama tüüpi objekte.*

4.2.2 Node.js

Node.js ei vasta täielikult püstitatud tingimustele, kuna Node.js ise on ainult platvorm, mille kaudu on võimalik servereid ehitada. Ta olemus on väga lähedane Java servleti komponentidele. Node.js platvormile on lugematu arv erinevaid REST teenuste loomiseks mõeldud raamistikke, kuid paraku on platvormi uudsuse tõttu üsna raske välja valida sellist, millel oleks väga ulatuslik ja tugev kogukond. Samas on selle platvormi eelis äärmiselt mitmekesine võimekus (Joyent, Inc, 2015) :

- asünkroonne, sündmustepõhine andmemudel
- takistustevaba sisend ja väljund (ei jää „lukku” mingi lahtise päringu tõttu)
- rakenduse kokkuehitamise aeg väga kiire, sisuliselt olematu

Platvormi ülesseadmine on üsna lihtne, vaja on paigaldada Node.js või io.js (Node.js baasil alternatiiv) laadides vastav pakk tootja kodulehelt. Lisamoodulite allalaadimine ja uuendamine käib käsurealt, kasutades Node.js-ga kaasnevat „npm” pakendihaldurit.

Kuna Node.js on veel üsna noor, siis ülekaalukalt populaarseid raamistikke temale veel välja kujunenud ei ole. Npm paketihalduri ametlikult kodulehelt statistikat uurides, selgub et eelpool käsitletud kriteeriume arvestavad populaarsemad REST teenuse arendamiseks saadaval olevad raamistikud on hapi (62000 allalaadimist eelmine kuu), sails (58000), koa

(36000), loopback (26000), actionhero (4600) (npm, Inc., 2015).

Toetus platvormile on kasvamas ja juba tänapäeval kasutavad seda mitmed suured firmad nagu eBay, PayPal, LinkedIn, Yahoo!, Microsoft, Uber, Wikipedia ja Netflix, Dow Jones (Joyent, Inc, 2015).

4.2.3 Spring Framework

Laialdaselt kasutuselolev raamistik, mis pakub äärmiselt palju funktsionaalsust juba baaskomplektis, sh REST teenuste arendamiseks. Keskkonda on võimalik väga lihtsalt püsti panna, kuna võimalik kaasa anda rakenduse server koos rakendusega. Lõplikule lahendusele on vaja lisada ainult andmebaasiühendus, kuid see on vajalik kõigi nimetatud raamistike puhul.

Toetab aspekt-orienteeritud lähenemist, mis tähendab, et annotatsioonidega on võimalik objektidele ja nende atribuutidele sisendada funktsionaalsust (Pivotal, 2015).

Java keele puhul on eelis heade IDE programmide, tööriistade saadavus ja platvormi laiaulatuslik kasutamine ning küpsus. Stack Overflow uuringust selgub, et Java on oma populaarsuselt 3. kohal (Stack Overflow, 2015).

4.3. Väljavalitud raamistik prototüübi tarbeks

Prototüübi valmistamiseks sai valitud Spring raamistik. Platvormi valimisel sai määravaks autori eelnev kokkupuude sellega. Kuigi sama kriteeriumi alusel oleks võinud valida ka PHP, ei soovinud autor tekitada keerulist ja operatsioonisüsteemist olenevat paigaldusjuhist. Node.js platvormi puhul jäi takistuseks vähene teadmine tehnilise platvormi tööst ja konkreetset raamistike ulatuslikest võimalustest.

5. Veebiteenuse koostamine

REST veebiteenuste alustalaks on kokkulepitud liidestamise standardid. (Fielding, 2000, 5.1, 5.1.5) See iga teenuspakkuja esmane ülesanne paika panna enda standardid. REST ei kirjelda teenuste väljundi spetsiifikat, vaid sõnab, et see peab olema ühtne. (Fielding, 2000, 5.2, 5.2.1) Selleks, et leida parim lahendus, analüüsis autor väga suurte firmade veebiteenuseid. Valitud firmade kriteeriumiteks oli lai kasutajaskond, sotsiaalne aspekt nende teenuses ja platvormi vanus (vähemalt 3 aastat). Valituteks osutusid Facebook, Twitter, Instagram ja SoundCloud. Olgu mainitud, et SoundCloudi veebiteenust kasutab Rada7.ee artistide lugude info saamiseks ka praegu.

Testimisel kasutati JSON-i andmeformaati. Lisaks valitud sotsiaalvõrgustikele uuritakse ka JSON-API standardit.

5.1. Sotsiaalvõrgustike veebiteenuste analüüs

Uurides erinevate teenuspakkujate REST dokumentatsiooni on selge, et nende standardid kattuvad ainult põhilistes ideedes:

- teenuse kasutamiseks peab registreerima endale konto ja seejärel looma ja siduma selle kontoga rakenduste kirje
- vastused peavad olema märgistatud õigete HTTP koodidega, kuigi osade koodide kasutamine ei ole igas teenuses sama semantilise tähendusega
- mõned teenused pakuvad veateate puhul lisaks enda metakoodi, mis ei ole HTTP kood, mõned (nt Twitter) lisavad enda poolt loodud HTTP koode
- tagastavad andmed peavad olema struktureeritud läbivalt samamoodi, ehk JSON-i, XML või mõne muu andmetüübi formaat peab alati vastama teatud struktuurile
- kõik võimalikud *endpoint*-id olid dokumenteeritud, kirjeldatud oli võimalikke parameetreid ja väljundeid.

5.1.1 Vastuste formaat

Kõigi puhul tavaks tagastada andmeid kas nimekirjana või üksiku objektina. Kui vastuseks oli

objekt millel saab olla mitu kirjet, näiteks kasutaja tehtud postitused, siis neid tagastati alati nimekirjana, isegi kui vastuseid oli ainult üks. See struktuur oli seotud *endpoint*-i semantikaga. Kui tulemusena ei saanudki tulla mitu kirjet, tagastati alati üks objekt aga kui tulemuses saab olla korraga mitu kirjet, siis oli vastus alati nimekirjana. Näiteks konkreetse kasutaja profiil oli üks kirje, kuid sõprade kirjed oli nimekirjana (näide 3).

```
{
    "id": 38645817,
    "id_str": "38645817",
    "name": "Kirill / Rada7.ee",
    "screen_name": "officialkirill",
    "location": "Tallinn, Estonia"
    // vastus lühendatud näite jaoks
}
```

```
{
    "users": [
        {
            "id": 1530873074,
            "id_str": "1530873074",
            "name": "Funderbeam",
            "screen_name": "funderbeam",
            "location": "Tallinn, Estonia"
        },
        {
            "id": 86440680,
            "id_str": "86440680",
            "name": "Annihilator",
            "screen_name": "annihilatorband",
            "location": "Ottawa, Ontario"
        }
    ],
    "next_cursor": 1437903611654361900,
    "next_cursor_str": "1437903611654361128",
    "previous_cursor": 0,
    "previous_cursor_str": "0"
    // vastus lühendatud näite jaoks
}
```

Näide 3. *Twitteri kasutaja profiili vastus on alati üks objekt kuid jälgijate vastus on nimekiri (isegi kui vastuses oleks ainult üks kirje).*

Instagrami puhul oli alati vastus „data” ülemobjekti sees (näide 4). See andis võimaluse lisada vastuse üldisele objektile muid metaandmeid, nagu näiteks vastuse kood või viide järgmise lehekülje REST päringu aadressile. Samas „data” objekti sees olevad andmed olid Instagrami teenusel samamoodi struktureeritud nagu teistel teenustel. Teised teenuspakkujad kasutasid „data” ülemobjekti sarnaseid lahendusi valikuliselt, näiteks Twitter tagastas ülemobjekte ainult nimekirjade puhul.

```
{
    "meta": {
        "code": 200
    },
    "data": {
        "username": "officialkirill",
        "website": "http://rada7.ee",
        "full_name": "Kirill",
        "counts": {
            "media": 29,
            "followed_by": 7,
            "follows": 10
        },
        "id": "431580715"
    }
    // vastus lühendatud näite jaoks
}
```

```
{
    "meta": {
        "code": 200
    },
    "data": [
        {
            "media_count": 457613,
            "name": "tallinn"
        },
        {
            "media_count": 3690,
            "name": "tallinnmusicweek"
        },
        {
            "media_count": 215,
            "name": "tallinnaülikool"
        }
    ]
    // vastus lühendatud näite jaoks
}
```


Näide 4. *Instagrami vastus „data” objektina: kasutaja profiili üksik kirje ja nimekiri siltidest.*

Iga teenuse puhul leidus *endpoint*-e, kust saadud vastused hõlmasid endas alamobjekte. Nende puhul jälgiti enamasti sama formaati, mis ka esmase objekti puhul.

Instagrami puhul oli iga alamobjekt „data” elemendi sees koos vastava objekti semantiliste lisadega. Näiteks kommentaari alamobjektiga oli kaasas nii kommentaarid ise, mis asusid „data” sees kui ka kommentaaride arv eraldi andmeväljana (näide 5). Sedasi ei pea kliendis tegema eraldi päringu, et kommentaarid kokku lugeda, isegi siis kui ollakse kuvamisel 3. lehekülje juures. Iga lehekülje päringuga tagastatakse kommentaaride koguarv.

```
{
    "data": [
        {
            "comments": {
                "count": 2,
                "data": [
                    {
                        "created_time": "1429265597",
                          "text": "Käisin ka seal ja ostsin kaks kommipakki a sa läksid veits
hulluks seal vist :D",
                        "id": "965072239610670145"
                    },
                    {
                        "created_time": "1429269322",
                        "text": "Jep :D varuks ikka sahtlisse",
                        "id": "965103482813394039"
                    }
                ]
            }
        }
    ]
    // vastust on lühendatud näite jaoks
}
```


Näide 5. *Instagrami pildi objekt koos kommentaari alamobjektiga. Kommentaari alamobjektis on ka valitud pildi kommentaaride arv kokku.*

Vigade puhul tagastati need enamasti „error” või „errors” objekti sees, kos veakirjeldusega ja veakoodiga. Facebooki puhul oli üllatav, et vea puhul tagastati koos vea objektiga HTTP koodiks 200, mis semantiliselt ei ütle masinale, et süsteemis tekkis viga.

5.1.2 Turvalisus

Kõik teenused nõudsid, et nende kasutamiseks omaks kasutaja kontot, mille alla kaudu saab tekitada rakendusi, kus igal rakendusel on oma ligipääsu identifikaator ja kood. Kumbki ei ole kasutaja poolt muudetavad, vajaduse või soovi korral saab süsteemil lasta ligipääsukoodid uuesti genereerida.

Twitter lubab enda teenuseid kasutada ainult OAuth autoriseerisstandardi kaudu, teised teenuspakkujad lubavad lihtsamaid päringuid teha lisades *endpoint*-i aadressile ainult oma identifikaator. OAuth on standardiseeritud autoriseerimismehanism, mille kaudu teenused saavad turvaliselt ühilduda mingi kolmanda osapoolega. Selle aluseks on kasutaja

identifikaator ja salakood, mille põhjal tehakse teenuspakkujale päring ning luuakse aeguv *access token*. Aegumisel peab *token*-i uuesti genereerima. Tegemist ei ole sessiooniga, sest erinevalt sessioonist päring ei pikenda *token*-i aegumise tähtaega (Hardt, 2014).

OAuth kaudu on võimalik ühildada kolmanda osapoole (nt SoundCloud) kasutajasüsteemi kaudu registreerimist oma rakenduses. Toimimismehanism on sama nagu tavaline OAuth päringu puhul, kuid süsteem peab vastava teenusepakkuja (SoundCloud) kasutaja andmed endale salvestama, et neid saaks igal sisenemisel võrrelda teenuspakkuja omadega (Sobers, 2015).

5.1.3 Alamressursid

Kõik teenuspakkujad sisaldasid ka alamressursse. Näiteks SoundCloudi kasutaja lugude kättesaamiseks peab tegema päringu aadressile /users/{id}/tracks. Sedasi on lihtsustatud konkreetse ressursiga seotud muude objektide kättesaamine.

5.2. JSON-API standard

JSON-API on nelja entusiasti poolt loodud dokument, mis kirjeldab kuidas peaks veebiteenus ülesehitatud olema. Muuhulgas kirjeldab JSON-API dokument kuidas tasub nimetada *endpoint*-e ja kuidas koostada dokumentatsiooni. Sarnaneb väga Instagrami teenuse struktuuriga.

JSON-API seab väga konkreetsed piirangud kuidas päring peab olema struktureeritud. Sealhulgas käsitleb see dokument põhjalikult vastuse formaati ning nõuab, et iga vastus asuks „data” objekti, vigade puhul „errors” sees (JSON-API, 2015). See on sarnane Instagrami formaadile, kuid JSON-API nõuab veel lisaks, et iga kirje juures oleks selle objekti metanimetus (näide 6).

```
{
    "data": {
        "type": "articles",
        "id": "1",
        "title": "Lorem ipsum"
        // muud artikli atribuudid
    }
}
```

Näide 6. *JSON-API päringu tulemus. Andmete metatüübiks on „articles”.*

Lisaks „data” objektile võib samal tasemel olla kaasas ka „meta”, „links”, „included” objektid (näide 7). „Meta” objekt kirjeldab tulemuse meta andmeid, näites artiklie puhul võib selleks olla litsensi info, autorite nimed. „Links” kirjeldab hüperlinke, mis on selle objektiga seotud, muuhulgas linki ressurssile. „Included” objekt sisaldab endas infot seotud lisaressursside kohta ja linke nendele. Spetsifikatsiooni järgi, on keelatud kliendil ja serveril muid kaasasolevaid andmeobjekte käsitleda (JSON-API, 2015).

```
{
    "meta": {
        "copyright": "Copyright 2015",
        "authors": [/*...*/]
    },
    "data": [{
        "type": "articles",
        "id": "1",
        "title": "Lorem ipsum",
        "links": {
            "self": "http://example.com/articles/1"
        }
    }],
    "links": {
        "self": "http://example.com/aricles/1"
    },
    "included": [{
        "type": "users",
        "name": "Real Person",
        "links": {
            "self": "http://example.com/people/9"
        }
    }, {
        "type": "comments",
        "id": "5",
        "body": "First!",
        "links": {
            "self": "http://example.com/comments/5"
        }
    }]
}
```

Näide 7. *JSON-API artikli ressursi päringu tulemus.*

6. Prototüüpimine

Prototüübi eesmärgiks on teenuse arendamise olulisemate sõlmpunktide läbiproovimine ja kitsaskohtade tuvastamine.

6.1. Turvalisus

Turvalisus on iga veebiteenuse alustala. Teenuse kolmandatele osapooltele pakkumisel on 2 levinud viisi: a) OAuth kaudu autoriseeirmine ja b) kliendi identifikaatorite kasutmine (identifikaatorite olemasolu on OAuth kasutamise eelduseks). Nende kaudu on võimalik pakkuda REST standardile vastavat oleku ülekandmist. Kuid see loob probleemi veebiteenuse organisatsiooni enda esmase veebikliendi kasutamisel. Olekuta teenus tähendab, et kasutaja sessiooni ei mäletata ning see omakorda tähendab, et pole võimalik pakkuda primaarses kliendis neid hüvesid millega tänapäeval kasutajad harjunud on. Näiteks „mäleta mind" funktsionaalsus, mille puhul kasutaja ei pea iga kord enda kasutajanime ja parooli sisestama. Või siis kui kasutaja viibib pikalt veebilehel, kirjutades midagi, siis selle postitamisel avastab ta, et tema sessioon on aegunud ning ta peab uuesti sisse logima. See omakorda tähendab, et tema postitatud sisu on läinud koos sessioniga kaotsi.

See tähendab, et sessioon on vaja siiski luua, kuid seda peaks piirama ainult organisatsiooni enda klientide tarbeks. Seejuures kuidas sessioon luuakse ei olegi oluline: kas selleks on OAuth kaudu autoriseerimine või enda ligipääsuandmete saatmine nagu tavalise sisselogimisvormi puhul. Selline lähenemine seab tehnilise raskuse rakenduse lähtekoodi poole pealt eristada olekuta ja olekuga seisu ning on kindlasti punkt mida peab veel lähemalt uurima. Prototüübis on ainult sessioonipõhine autentimine, kuid rakenduse lihtsuse tõttu on enamus andmeid kättesaadavad ilma autentimiseta.

6.2. Päringud

REST standard ei tee ettekirjutusi päringute vastustele. Selle tõttu on kohustus arendajatel välja töötada võimalikult täpne vastuste formaat, mis oleks paindlik ja hästi masinloetav. Prototüübis kasutatakse JSON-API formaadile sarnast ülesehitust. HTTP koodide määramisel on võimalik seda erinevalt teha, kuid nende olemus peab siiski võimalikult täpselt kirjeldama

vastuse seisu.

6.3. Teenuse versioonid

Teenuse versioonide uuendamine on üks keerulisemaid aspekte. Väljastades mingi versiooni on organisatsioonil kohustus hoida see versioon täpselt sellisena nagu ta on ja mitte seda drastiliselt muuta. See käib nii aadressite kohta kui ka väljundi formaadi kohta. Muutes enda teenuse arhitektuuri tekitab see teenuse teistele tarbijatele olukorra, kus nad ei jõua muudatustega kohe kaasa tulla. See tähendab, et uue versiooni väljastamisel peaks organisatsioon hoidma alles ka vana teenuse (Fielding, 2000, 4.1.4.2). Samas, ei pea seda lõputult käigus hoidma, kuid üleminekuperioodi ajaks on selle kättesaadavus tähtis ja hea tahte märk.

Teenuste väljatöötamisel on seega äärmiselt oluline, et teenus oleks põhjalikult läbi mõeldud enne tema lõplikku rakendamist. Kiirelt areneva platvormi puhul, nagu Facebook esineb selliseid muudatusi päris tihti, ning nende teenuses on tavaline versioonide suhteliselt lühike kestvus.

6.4. Uuele arhitektuurile kolimine

Selleks, et kolida Rada7.ee uuele teenuspõhisele arhitektuurile tuleb luua klient (*Single Page Application*) kui ka veebiteenus. Klient võib töötada ka praeguses Apache HTTP serveris edasi, kuid veebiteenus tuleb Node.js või Java platvormi puhul tööle panna eraldi serveris. Kuna www.rada7.ee serveri konfiguratsioonis ei muutu midagi, vaid sellel aadressil töötav rakendus asendatakse *Single Page Application*-iga, peaks teenuse tehniline üleminek olema üsna sujuv. SPA peab tagama täpselt samasuguste teemade, artiklite, uudiste, loosimiste, ürituste, artistide ja alamfoorumite aadressite olemasolu nagu praegune veebirakendus, et kõik vanad lingid töötaksid edasi.

JavaScripti põhise kliendi puhul peab lisaks arvestama, et lehe sisu oleks otsirobotitele kättesaadav. Tänapäeval veel kõik otsirobotid ei suuda indekseerida Javascripti poolt genereeritud tulemust, kuna nad eeldavad serveri poolset vastust HTML-ina. SPA puhul ei tagastata serverist HTML-i vaid genereeritakse see veebilehitsejas. Prototüüp ei käsitle lahendust sellele probleemile.

Kokkuvõte

Bakalaureusetöö eesmärgiks oli koostada teekaart Rada7.ee veebiteenuse edasiseks arendamiseks. Selleks uuris autor erinevaid sotsiaalmeedia veebiteenuseid ja leidis nende ühisosasid, millest võiks enda veebiteenuse koostamisel õppust võtta. Masinatevahelise suhtluse aluseks on üksteisest arusaamine ning andmete masinloetavus. See nõuab rangete formaatide väljatöötamist ja nendest kinni pidamist. Formaadid ei pea olema ülemaailmselt standardiseeritud, piisab kui organisatsioon loob dokumendi, kus kirjeldab enda teenuse standardeid ning hoiab neid teenuseüleselt ühtsena.

Internet on olemuselt püsimatu süsteem ja analüüsides suurtemate sotsiaalmeedia teenuse dokumentatsioone on selge, et sellise arhitektuurini ei jõutud korraga, vaid see arenes ajapikku. Siit võib ka järeldada, et teenuse loomisel ei peaks olema eesmärk korraga suure platvormi valmistamine, vaid edasiliikumine peab toimuma iteratiivselt.

Analüüsi käigus selgus, et kasutajate õiguste lahendamiseks on põhiprobleem kasutaja autoriseerimisel ja autentimisel, mitte raamistikusiseses kasutajaõiguste haldamises. Rada7.ee puhul ei saa õiguste süsteem baseeruda rollidel, kus rolli olemasolul lubatakse mingi tegevus või keelatakse. Sellise süsteemi puhul kontrollitakse kasutaja ligipääsu ressursile. Kui see on olemas, siis antakse vastuseks küsitud objekt, kui puudu, siis vastuseks tagastatakse palve end autentida. Rada7.ee puhul on enamasti ressurssi ligipääs kõigile kasutajatele sama, kuid kasutajad saavad eritasemeliselt andmetele ligi. Autor eeldas, et see on süsteemi üks keerulisemaid osasid, kuid analüüsi käigus selgus, et keerulisem on hoopis õiguste hindamine ja määramine masinsuhtluses. See on kindlasti punkt mis vajab katsetamist ja edasiuurimist.

Alustama peaks lihtsast teenusest, millele arendatakse ajapikku lisavõimalusi. Esimeseks teenust tarbivaks kliendiks peaks saama Rada7.ee Single Page Application klient. Silmas tuleb pidada, et teenus ei saaks liiga ühe kliendi spetsiifiliseks.

Tehniline lahendus andmete kuvamiseks ja sisendi töötlemiseks ei ole keeruline kui kindel formaat on paigas. Enne platvormi avalikustamist, peaks veebiteenus küpsema ja töötama mõnda aega ainult üksiku kliendiga.

Summary

Title: Development Roadmap of a Web Service: the Case of Rada7.ee

The objective of this thesis is to map the necessary components for developing a REST-architecture based web service for Rada7.ee. To achieve this purpose, several existing web services of popular social networks were chosen and analyzed. Representational state transfer or REST, is a HTTP based web service standard, that does not store user's state. One of the key objectives of REST is to provide a unified interface for consuming a service. As such, the aim was to find any common ground among existing popular services and draw from their experience.

A smaller task was to compare different frameworks that would help achieve a technical solution for creating a machine consumable web service.

To give an example of how all of this should work together, a small prototype was also created.

The analysis showed that there are no unified standards among the selected web services. Each vendor provided its own standardization in returning objects and accepting input. They exhibit common design patterns, in the way how single and multiple objects are delivered, accepted, and how errors are served, but their formats' semantic structure is different. Among them, JSON-API standard was also looked at. I has not gained a widespread acceptance yet, although Instagram, one of the web services that was analyzed, did in fact use a lot of ideas proposed in the JSON-API draft.

The security for such a web service must be diverse. This is due to the fact that it should also be able to serve a session based single page application client, hosted from the main website www.rada7.ee. The security layer should be able to handle website specific needs and keep them clear from the other parts of the service that are stateless, as per REST specification. This architectural division should be achieved via programming the service in such a fashion that both parts are kept separate.

Further objectives should be creating a standardized document describing the services' semantics and researching implementing security around the aforementioned limitation.

Kasutatud kirjandus

Alexa. How popular is stackoverflow.com? Kasutamise kuupäev 3. mai 2015.a., allikas http://www.alexa.com/siteinfo/stackoverflow.com

ECMA International. Introducing JSON. Kasutamise kuupäev 3. mai 2015.a., allikas http://www.json.org/

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irving. Kasutamise kuupäev 3. mai 2015.a., allikas http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fielding, R., & Reschke (2014), J. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. Kasutamise kuupäev 3. mai 2015.a., allikas http://tools.ietf.org/html/rfc7231#section-4.3

Hardt, D (2014). The OAuth 2.0 Authorization Framework. Kasutamise kuupäev 3. mai 2015.a., allikas http://tools.ietf.org/html/rfc6749

HashiCorp. Why Vagrant? Kasutamise kuupäev 3. mai 2015.a., allikas http://docs.vagrantup.com/v2/why-vagrant/

Internet Engineering Task Force. (2011). The WebSocket Protocol. Internet Engineering Task Force. Kasutamise kuupäev 3. mai 2015.a., allikas https://tools.ietf.org/html/rfc6455

Joyent, Inc. About Node.js. Kasutamise kuupäev 3. mai 2015.a., allikas https://nodejs.org/about/

Joyent, Inc. Industry. Kasutamise kuupäev 3. mai 2015.a., allikas https://nodejs.org/industry/

JSON-API. JSON-API format. Kasutamise kuupäev 3. mai 2015.a., allikas http://jsonapi.org/format/

Kippar, J. (2009). *XML rakendused*. Tallinna Ülikool, Tallinn. Kasutamise kuupäev 3. mai

2015.a., allikas http://minitorn.tlu.ee/~jaagup/kool/java/loeng/xmlrak/xmlrak.pdf

Milovidov, K. (2015). *JavaScripti haldur veebirakenduses* (seminaritöö). Tallinna Ülikool.

npm, Inc. npm package manager. Kasutamise kuupäev 3. mai 2015.a., allikas https://www.npmjs.com/

Nurme, A. (2006). *Tarkvaraarendusprojekt Rada7 näitel*. Tallinna Ülikool, Tallinn.

PHP Framework Interop Group. PHP Framework Interop Group. Kasutamise kuupäev 3. mai 2015.a., allikas http://www.php-fig.org/

Pivotal. Aspect Oriented Programming with Spring. Kasutamise kuupäev 3. mai 2015.a., allikas http://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html

Sobers, R. Introduction to OAuth (in plain English). Kasutamise kuupäev 3. mai 2015.a., allikas http://blog.varonis.com/introduction-to-OAuth/

Stack Overflow. 2015 Developer Survey. Kasutamise kuupäev 3. mai 2015.a., allikas http://stackoverflow.com/research/developer-survey-2015

Stackshare. Node.js Stackshare. Kasutamise kuupäev 3. mai 2015.a., allikas http://stackshare.io/node-js

Symfony. 10 criteria for choosing the correct framework. Kasutamise kuupäev 3. mai 2015.a., allikas http://symfony.com/ten-criteria

Takada, M. Single page apps in depth. Kasutamise kuupäev 3. mai 2015.a., allikas http://singlepageappbook.com/goal.html

The World Wide Web Consortium. (2004). Web Services Architecture. The World Wide Web Consortium. Kasutamise kuupäev 3. mai 2015.a., allikas http://www.w3.org/TR/ws-arch

The World Wide Web Consortium. (2004). Web Services Glossary. The World Wide Web Consortium. Kasutamise kuupäev 3. mai 2015.a., allikas http://www.w3.org/TR/ws-gloss/

Lisad

Lisa 1: Mõisted

Selles peatükis kirjeldatakse bakalaureusetöös kasutatud mõisteid.

- Klient – sügavalt kontekstist sõltuv mõiste, võib tähendada veebilehitsejat kui silmas peetakse kasutaja ja internetilehekülgede vahelist interaktsiooni, kuid võib olla ka mingi masin, mis suhtleb teenusega. Masinaks võib olla ka mingi muu kolmanda osapoole teenus. Üldiselt võib klienti kirjeldata kui „teenuse tarbija”.
- Kasutaja – masin või inimene, mis tarbib teenust
- URL – lühend sõnadest *Uniform Resource Locator*, üldine mõiste mis mida kasutakse internetis levivate objektide/ressursside kättesaamiseks, veebiaadress.
- *endpoint* – olemuselt sama, mis URL kuid kasutatakse veebiteenuste kontekstis
- *slug* – unikaalne ja inimloetav mingitest sõnadest (eelkõige objekti pealkiri) kokku pandud objekt, koosneb numbritest, ladina tähestikust ja sidekriipsudest (- märk). Kõik mitteladinakeelsed tähed asendatakse mõne ladinakeelse tähega. Näiteks ä = a, é = e, õ = o, ü = u. Keelte puhul, kus ladina tähti üldse ei kasutata (nt vene, jaapani), võib kasutada transliteratsiooni *slug*i koostamiseks. Kõik tühikud ja muud märgid asendatakse sidekriipsudega.
- *cookie* – veebilehitseja kaudu salvestatud fail, kus on kirjas mingi informatsioon, mida on võimalik veebilehitsejast veebiserverisse saata ja sealt lugeda.
- *stack* – tehnoloogiate hulk, mis on mõeldud koos kasutamiseks
- bänner – reklaami jaoks kasutatav pilt või graafika
- *full text search* – täistekst otsing, otsingu tüüp kus otsitakse teksti sisust, mitte metaandmetest
- *convention over configuration* – tarkvaraarenduse lähenemine, mis eelistab kindlat konventsiooni koodi kirjutamisel, kindlaks määramata konfigureerimise asemel

- *dependency injection* – koodimuster, kus sõltuvusi teistele objektidele delegeeritakse mingi ülema vooga
- kasutajaliides – inimese ja masinavahelise interaktsiooni kokkupuutekoht
- *serializiation* – protsess, millega muudetakse andmestruktuurid talletavasse formaati ning mida on hiljem võimalik taasesitada orginaalstruktuurina
- *token* – eesti keeles oleks lähim mõiste ühekordne pilet
- portimine – mingi rakenduse koodi ületoomine ühest keskkonnast teise

Lisa 2: Prototüüp

Prototüüp on kättesaadav aadressilt https://bitbucket.org/officialkirill/rada7-proto/. Prototüüp nõuab käivitamiseks Java 8 (JDK) olemasolu ja selle toimimist käsurealt. Java 8 saab alla laadida Oracle kodulehelt: http://www.oracle.com/technetwork/java/javase/downloads/. Samal lehel on võimalik leida ka juhised Java paigaldamiseks.

Rakenduse paigaldamiseks on vaja see eelnevalt alla laadida ning paigaldada selle sisu kusagile kausta. Seejärel tuleb käsurealt liikuda sinna kausta ning käivitada käsk „gradlew” (Windows) või „sh gradlew” (Linux/MacOSX).

Kokkuehitamiseks kasutatakse Gradle-it, mis esmasel laadimisel laeb alla kõik vajalikud lisapaketid. Selle tegevuse peale kulub umbes 5-10 minutit, olenevat kättesaadavast internetikiirusest.

Kokkuehitamise lõpus käivitab Gradle serveri, mis on kättesaadav aadressilt http://localhost:8080 sellest arvutist, kus rakendus käivitati. Eduka käivitamise puhul ilmub käsureale kiri `org.rada7.prototype.Application : Started Application in 13.706 seconds (JVM running for 14.961)`.

Serveri sulgemiseks tuleb konsoolis vajutada Ctrl + C või siis sulgeda konsooliaken.