TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

ITV40LT
Oliver Eivak 134917

PILLIÕPPE JÄLGIMISE VEEBIRAKENDUSE ARENDAMINE

Bakalaureusetöö

Juhendaja: Jaagup Irve
Magister
Tarkvarainsener

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Oliver Eivak

18.05.2016

Annotatsioon

Selle bakalaureusetöö eesmärgiks on luua muusikainstrumentide harjutamise üles märkimise jaoks digitaalne lahendus ning siduda see Pomodoro tehnikaga. Töö tulemus võiks pakkuda huvi paljudele muusikahuvilistele, kes soovivad saada oma harjutamisest paremat ülevaadet.

Töö käigus luuakse veebirakendus, mis lubab kasutajatel lisada lugusid neid nime järgi otsides, üles märkida harjutamist ja näha ülevaateid ning töötab erinevate suurustega ekraanidel. Rõhku pannakse ka protsessile, kirjutatakse teste ja seatakse üles pideva integratsiooni server.

Töö kirjeldab rakenduse arhitektuuri, kasutatud tehnoloogiaid, integratsioone väliste süsteemidega. Mõningate tehniliste probleemide lahendamist vaadeldakse lähemalt. Töö tulemuseks on reaalselt töötav rakendus, mis täidab seatud eesmärke.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 44 leheküljel, 5 peatükki, 10 joonist, 8 koodinäidet, 1 tabelit.

Abstract

Development of an Instrument Practice Tracking Web Application

The main aim of this thesis is to develop a digital solution to keeping track of musical instrument practice that also implements the Pomodoro technique. The solution could be of interest to anyone who practices a musical instrument and would like have a better overview.

The solution is made as a web application that works well in modern desktop, tablet and mobile browsers. The application allows users to authenticate, add songs by searching by name, track practice sessions and see overviews over different time frames.

Importance is not only placed on the final solution but also on the process itself. The thesis describes both the functional and nonfunctional requirements for the application. From these the project backlog is created, which consists of prioritized user stories. Throughout development the backlog is updated and kept prioritized to make sure important features are finished in the limited time frame available. A test server along with continuos integration software is set up in the beginning to automatically retrieve code from the repository, build, test and deploy it.

The thesis describes the arhitecture of the system, the main technologies used and integrations with external systems. Some technical problems are viewed in more detail. The result of this thesis is a real, working and usable application that fulfills its purpose of tracking practice sessions and giving overviews.

The thesis is in Estonian and contains 44 pages of text, 5 chapters, 10 figures, 8 code examples, 1 table.

Lühendite ja mõistete sõnastik

Pomodoro	Kindla pikkusega sessioon, tavaliselt 25 minutit. Selle töö raames on Pomodoro all mõeldud ühte harjutussessiooni.
JPA	*Java Persistence API* – standardne liides relatsioonilste andmete haldamise jaoks Java platvormil.
DAO	*Data Access Object* on objekt, mis võimaldab suhtlust mõne andmete säilitamise meetodiga, näiteks andmebaasiga.
YAML	*YAML Ain't Markup Language* on inimloetav andmete serialiseerimise keel

Sisukord

1 Sissejuhatus 11
1.1 Eesmärgid 11
1.2 Metoodika 11
2 Analüüs 13
2.1 Funktsionaalsed nõuded 13
2.2 Mittefunktsionaalsed nõuded 14
2.3 Töö planeerimine 14
2.4 Olemi-suhte diagramm 15
3 Arendus 17
3.1 Tehnoloogiline ülevaade 17
3.1.1 Back-end 17
3.1.2 Front-end 20
3.2 Arhitektuur 22
3.3 REST liides 24
3.4 Liidestused teiste süsteemidega 24
3.5 Tehniliste probleemide lahendamine 26
3.5.1 Transaktsioonide haldus 26
3.5.2 Autentimine rakenduse kasutamisel 27
3.5.3 Identifikaatorite peitmine 28
3.5.4 Konfiguratsioon 29
3.5.5 Animatsioonid 30
3.6 Testimine 31
3.6.1 Unit testid 31
3.6.2 Integratsioonitestid 34
3.7 Pidev integratsioon 34
4 Hinnang tulemusele 36
4.1 Mobiilivaadete analüüs 38
4.2 Probleemid 40
5 Kokkuvõte 41

6 Kasutatud kirjandus 42
Lisa 1 – Sessiooni ID genereerimise koodinäide 45
Lisa 2 – Harjutusvaade 46
Lisa 3 – Mobiilivaated 47

Jooniste loetelu

Joonis 1. Olemi-suhte diagramm. 16
Joonis 2. Rakenduse komponentdiagramm. 23
Joonis 3. Paigaldusdiagramm. 23
Joonis 4. Koodi testidega kaetus..... 31
Joonis 5. Projekti punktide jäägi progress. 36
Joonis 6. Punktide jagunemine iteratsioonidesse. 37
Joonis 7. Kasutajalugude tüüpide jaotus. 37
Joonis 8. Harjutusvaade tahvelarvutil. 38
Joonis 9. Harjtusvaade mobiilivaates. 39
Joonis 10. Navigatsioon mobiilivaates. 40

Koodinäidete loetelu

Kood 1. Lombok annotatsioonide näide. 20
Kood 2. RestModule koodinäide. 26
Kood 3. Autentimispäringu näide. 27
Kood 4. Autentimispäringu vastuse näide. 27
Kood 5. Identifikaatorite töötlemise näide lihtsustatud Song objekti põhjal. 29
Kood 6. Back-end rakenduse konfiguratsiooniklass. 30
Kood 7. DAO klassi testimine. 32
Kood 8. Testimine mock objektidega. 33

Tabelite loetelu

Tabel 1. REST liidese ülevaade..... 24

1 Sissejuhatus

Paberi ja pliiatsiga instrumendi harjutamise ülesmärkimine on võimalik, aga mitte eriti otstarbekas. Digitaalne lahendus võiks olla mugavam ja pakkuda paremaid ülevaateid. Selline rakendus võiks pakkuda huvi kõigile, kes harjutavad mõnda muusikainstrumenti.

1.1 Eesmärgid

Töö esimeseks eesmärgiks on luua muusikainstrumendi harjutamise jälgimiseks digitaalne lahendus. Keskkonnas peaks saama üles märkida lugude harjutamist ja näha kokkuvõtteid ning tuleks integreerida mõne välise teenusega, et saada harjutatavate lugude metaandmeid (näiteks esitaja). Keskkond peaks olema kasutatav nii lauaarvutis, tahvelarvutil kui mobiilil.

Töö teiseks eesmärgiks on viia kokku pilliharjutamine ja Pomodoro tehnika, kus tegutsetakse järjest ja keskendunult 25 minuti ning seejärel tehakse 5 minutit pausi.

Kolmandaks eesmärgiks on viia arendus läbi nii, et saadud tarkvara oleks kergesti edasi arendatav ja laiendatav uute funktsionaalsustega. Ei tohiks tekkida tehnilist võlga, tarkvara õigesti töötamine (vähemalt serverirakenduse poolel) peaks olema automaatselt kontrollitav.

1.2 Metoodika

Vastavat nõuetele kirjeldatakse projektihalduskeskkonnas kasutajalood ja moodustatakse *backlog* – prioritiseeritud funktsionaalsuste nimekiri. Kogu funktsionaalsuse kirja panemine ei ole alguses tähtis, küll aga on vajalik *backlog*-i pidev prioritiseerimine, et oluliseimad funktsionaalsused mahuksid ära piiratud arendusaja sisse.

Arenduse alguses seatakse üles testserver, millele paigaldatakse pideva integratsiooni tarkvara, mis hakkab automaatselt arendatava rakenduse koodi koodihoidlast alla laadima, testima ja uusi versioone testserverile paigaldama.

Rakendus ehitatakse kihilise arhitektuuriga ning Java ja Javascripti abil.

2 Analüüs

Selles peatükis kirjeldatakse loodava rakenduse funktsionaalsed ja mittefunktsionaalsed nõuded, selgitatakse töö planeerimist ning tuuakse välja olemi-suhte diagramm.

2.1 Funktsionaalsed nõuded

1. Kasutaja saab end registreerida.
2. Kasutaja saab autentida kasutajanime ja parooliga, Google ja Facebook kontoga.
3. Kasutaja saab välja logida.
4. Süsteemil on päis, millel on menüü.
5. Saab lisada uue loo sisestades ainult loo nime.
6. Saab otsida uut lugu, mida lisada, välise teenuse abil.
7. Süsteem salvestab loo metaandmed välisest teenusest.
8. Süsteem lubab lisada lugusid ka metaandmeteta.
9. Süsteem võimaldab lugusid kustutada.
10. Süsteem näitab kasutaja lisatud lugude nimekirja koos metaandmetega.
11. Kasutaja saab alustada loo harjutamist ja näha kui palju aega on veel jäänud harjutada.
12. Harjutussessiooni lõpus lisatakse loole käesoleva päeva alla uus pomodoro.
13. Kasutaja saab katkestada harjutamise.
14. Süsteem näitab iga loo juures lisaks tekstilistele metaandmetele ka albumi, millel lugu asub, pilti.
15. Kasutaja saab lisada pomodorosid ka tagantjärgi.
16. Kasutaja saab kustutada pomodorosid.
17. Kasutaja saab muuta pomodoro pikkust.
18. Igal lool on ka oma lehekülg eraldi aadressiga.
19. Loole saab lisada väliseid linke.
20. Kui väline link on youtube video, siis näidatakse seda lehe sees (embedded).
21. Kui lool on mitu youtube video linki, saab kasutaja valida, millist hetkel vaadata.

22. Kui lool on youtube video link, näidatakse seda lugude nimekirja vaates albumi pildi asemel.
23. Hetkel harjutatav lugu peab olema nähtav iga lehe peal.
24. Kasutaja saab näha harjutamise kokkuvõtet, mis koosneb lugude nimekirjast ja selle loo harjutussessioonidest (pomodorodest) mingi kindla ajavahemiku jaoks.
25. Ajavahemikest on võimalik valida nädal, kuu, aasta ja kõik.
26. Kasutaja saab liikuda ajavahemike puhul edasi ja tagasi – näiteks vaadata eelmist nädalat, üleelmist jne.
27. Süsteemil on avaleht, kus autentimata kasutajatele kuvatakse selle tutvustus.
28. Avalehel on näha populaarseimad artistid – need kelle lugusid on kõige rohkem harjutatud.
29. Kasutaja näeb nimekirja artistidest, kelle lood võiksid talle huvi pakkuda (vastavalt sellele, milliseid lugusid on kasutaja seni harjutanud)

2.2 Mittefunktsionaalsed nõuded

1. Süsteemi kasutajaliides on inglisekeelne ja ei pea olema tõlgitav.
2. Süsteemi kood on inglisekeelne.
3. Süsteem salvestab logisid faili, iga päeva jaoks on uus fail. Kindlasti peavad olema logitud veateated.
4. Süsteem peab olema kasutav ka mobiilil ja tahvelarvutil - disain peab olema *responsive*.
5. Kujundus peab rakendama Google Material Design-i.
6. Süsteem töötab veebilehitsejate Google Chrome ja Mozilla Firefox uusimate versioonidega.
7. Süsteem peab töötama vabavaralisel operatsioonisüstemil.
8. Kuna kasutaja saab sisestada harjutussessioone ka tagantjärgi, ei ole süsteemi kättesaadavus väga suure tähtsusega.

2.3 Töö planeerimine

Vastavalt nõuetele said loodud kasutajalood ja need suuremate funktsionaalsuste järgi gruppidesse (*epic*) organiseeritud. Selleks on kasutusel projektihalduskeskkond Pivotal Tracker. Kokku sai loodud 16 gruppi:

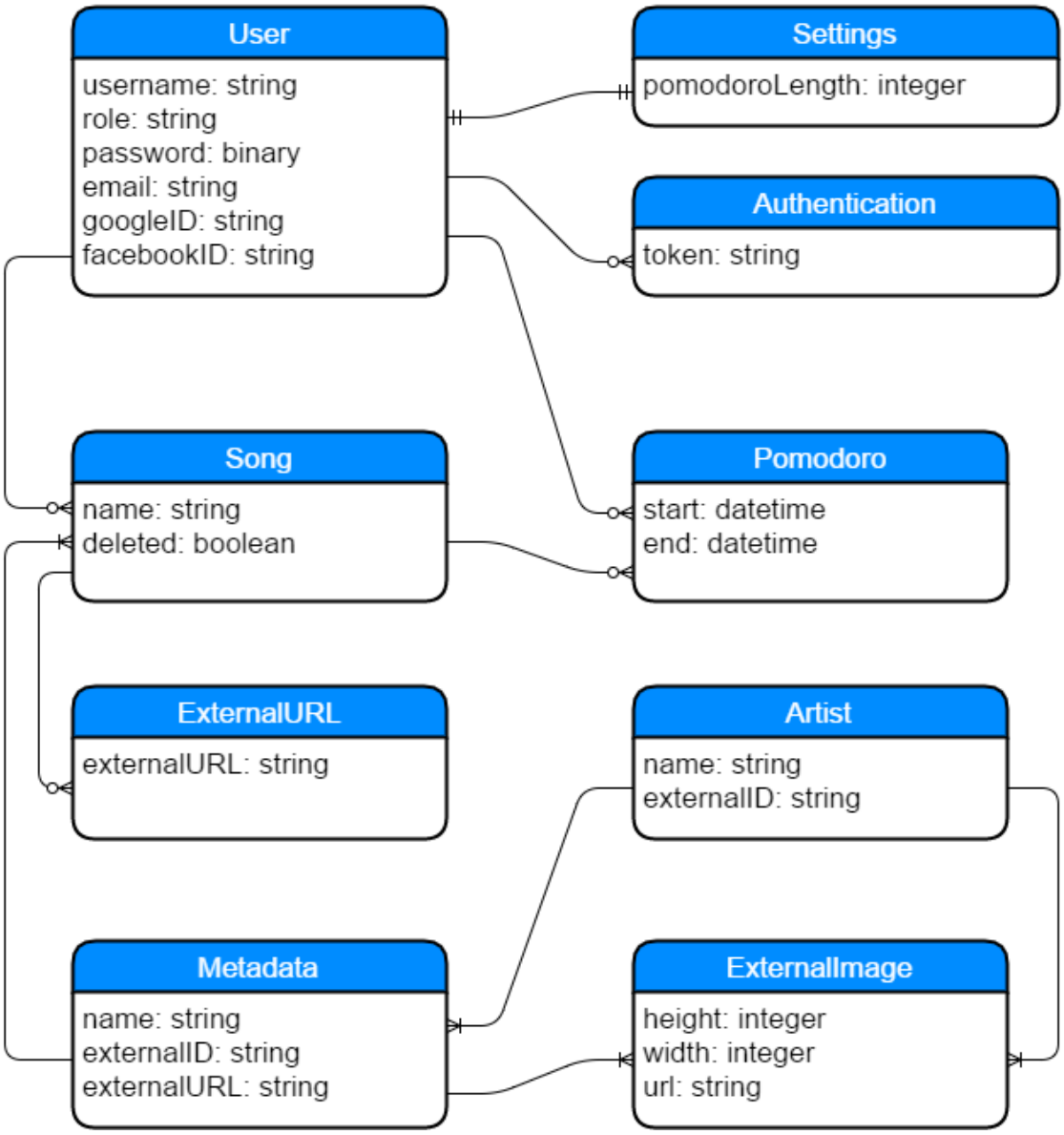
- registeerimine
- sisse logimine
- välja logimine
- päis
- harjutusvaade
- uue loo lisamine
- harjutamine
- seaded
- laulu detailvaade
- ajalõigud (nädal/kuu/aasta/kõik)
- lingi lisamine
- esileht
- loo kustutamine
- Google-iga sisse logimine
- Facebook-iga sisse logimine
- soovitused

Kuna loodud sai üle 100 kasutajaloo, siis neid kõiki siin eraldi välja ei tooda, küll aga on toodud näiteks uue loo lisamise *epic* koos oma kasutajalugudega, et anda aimdus sellest, millised need tavaliselt on:

- Kasutaja tahab sisestada uue loo nime.
- Kasutaja tahab nupule vajutades loo lisada.
- Kasutaja tahab loo nime kirjutades näha otsingutulemusi.
- Kasutaja tahab otsingutulemustes näha lugude nimesid.
- Kasutaja tahab otsingutulemustes näha lugude esitajaid.
- Kasutaja tahab otsingutulemustes näha lugude pilte.
- Kasutaja tahab lisada uue loo klikkides loole otsingutulemustes.

2.4 Olemi-suhte diagramm

Järgnevalt on joonisel nr. 1 toodud ära süsteemi olemi-suhte diagramm. Süsteemi põhilisteks objektideks on kasutaja, lugu, ja pomodoro.



Joonis 1. Olemi-suhte diagramm.

3 Arendus

3.1 Tehnoloogiline ülevaade

Järgnev ülevaade on jaotatud kahte loogilisse ossa – back-end ja front-end projekt.

3.1.1 Back-end

Back-end on kirjutatud **Java 8**-s. Kuna tegemist on täiesti uue projektiga ning selle jooksutamise keskkond on vabalt konfigureeritav, siis ei ole ühtegi põhjust, miks kasutada vanemat versiooni.

Maven on tööriist Java projektide ehitamiseks ja haldamiseks. Maven võimaldab kirjeldada projekti sõltuvusi pom failis. POM on lühend Project Object Model-ist ehk projekti objektmudel. Samuti pakub Maven kõigile projektidele ühesugust ehitamise süsteemi [1].

Projekti pom failis antakse projektile nimi ja versioon, sõltuvused ning vajadusel projekti ehitamise seaded. Maven laeb alla kõik projekti sõltuvused, sõltuvuste sõltuvused jne [2]. Sõltuvused võivad olla nii kohalikus masinas, mõnes artefaktide serveris või (tavaliselt) keskses serveris (Maven 2 Central Repository [3]).

Maveni kasutamine dikteerib ka algupärase projekti struktuuri. On kaustad *src* ja *target*, *src* kaust jaguneb *main* ja *test* kaustaks, mõlemad jagunevad *java* ja *resources* kaustaks. Arusaadavalt tähistab *src* lähtekoodi kausta, *main* alla jääb põhiline programmi kood ning *test* kausta testid. *Java* all on programmi kood, *resources* kaustas muud failid.

Gwizard on java teek, mille eesmärgiks on teha lihtsaks REST stiilis töötavate veebirakenduste loomine. Seda ei saa nimetada raamistikuks, nagu seda on näiteks Spring, kuna see on väiksem ja ei vaja palju spetsiifilisi teadmisi enda kohta. Gwizard on pigem natuke liimi erinevate teekide vahel, et nendega arendamise alustamine läheks valutult [4].

Gwizard kasutab ka teeke Dropwizardist, mis on raamistik veebirakenduste loomiseks. Suurim erinevus on see, et Gwizard kasutab Guice-i, Dropwizardis puudub aga sõltuvuste süstimine. Gwizard tekkis, kui arendajad lisasid Dropwizardile Guice-i ja hakkasid seda jupp haaval asendama ning üleliigset koodi vähendama. Gwizard ei ole siiski Dropwizardi edasiarendus, vaid uus teek, mis lihtsalt kasutab osasid olemasolevaid Dropwizardi teeke [4].

Gwizard koosneb erinevatest maven moodulitest, lubades nii valida just need funktsionaalsused, mida projektis vajatakse. Olemas on moodulid konfiguratisooni, logimise, servlet-ide, RESTEasy, Jersey, Hibernate, Guava teenuste, meetrika, teenuste tervise kontrolli (*health checks*) ja rpc jaoks [4].

Guice on sõltuvuste süstimise raamistik. Guice vähendab vajadust kasutada vabrikuid (*factory*) ja *new* võtmesõna, nende asemel on @Inject annotatsioon, millega saab küsida Guice-i käest objekte. Guice-i kasutamise tulemuseks on kergemini hallatav, taaskasutatav ja testitav kood [5].

Objektide ehitamiseks on vaja nende sõltuvusi, sõltuvuste ehitamiseks aga nende oma sõltuvusi jne, ehk objekti ehitamiseks on vajalik objekti graaf. Guice-is on objekti ehitajaks *injector*. Injector-i loomise aluseks on moodulid, mis kirjeldavad seoseid klasside vahel, näiteks milline klass implementeerib millist liidest või milline klass pakub mingit tüüpi klassi objekte (Provider klassid) [6]. Siis on võimalik süstida koodi objekte viidates vaid liidesele ja sõltuvust vajav kood ei pea teadma, kes seda liidest tegelikult implementeerib.

Guice toetab sõltuvuste süstimist konstruktori kaudu, lisades @Inject annotatsiooni konstruktorile ja nimetades kõik vajalikud sõltuvused kui kontruktori parameetrid, meetodide kaudu, kui lisada @Inject meetodi kohale, aga ka väljade süstimist, kui lisada annotatsioon iga vastava klassi välja kohale, mida soovitakse saada Guice-i käest objekti loomisel [7].

Hibernate on objekt-relatsioonilise vastendamise raamistik ning Java Persistence API (JPA) implementatsioon (st. arendajad ei pea kirjutama käsitsi Java Database Connectivity koodi). Hibernate võimaldab kirjutada rakendusi, mille andmed asuvad relatsioonilises andmebaasis. Selleks, et muuta klass püsivaks, tuleb vaid lisada @Entity annotatsioon, ei ole vaja laiendada mingeid teisi klasse või implementeerida liideseid.

Hibernate ei vaja spetsiaalseid andmebaasi tabeleid või välju ja suudab kiiruse parendamiseks genereerida palju SQL lauseid juba süsteemi käivitamisel valmis [8].

Hibernate tegeleb sellega, et panna omavahel vastama andmed Java klassidest ja andmebaasi tabelitest, hoides nii kokku arendajate aega, mis kuluks muidu tavalise SQL ja JDBC koodi kirjutamisele. Küll aga saab vajadusel kasutada SQLi võimalusi [9].

MySQL on Oracle poolt arendatav SQL andmebaasi halduse süstem, mida rakendus Hibernate abil kasutab. Sama hästi oleks võinud kasutada ka MariaDB andmebaasi, aga käesoleva rakenduse jaoks ei ole eriti oluline, milline on andmebaasitarkvara arendusprotsess või kui palju erinevaid salvestusmootoreid pakutakse.

Guice persist on teek, mis lihtsustab teiste püsivandmete teekide (nt JPA) kasutamist**.** Guice Persist lubab kergemalt tegeleda transaktsioonide ja tööühikutega (units of work). [10] Käesolevas rakenduses aitab selle kasutamine vähendada *boilerplate* koodi.

Jackson on JSON-i parsimise teek. Jackson suudab luua Java objektidest JSON objekte ja vastupidi, mis on veebirakenduste jaoks väga kasulik. Jackson ei ole piiratud vaid JSON-iga, vaid suudab töödelda ka näiteks CSV, XML, YAML ja muid formaate. [11]

RESTEasy on JAX-RS implementatsioon, millega saab ehitada REST teenuseid ja selle arendajaks on Jboss. RESTEasy-t on võimalik kasutada koos Jacksoniga, et päringuid ja vastuseid töödelda, samuti Gwizardi abil integreerida seda Guice-iga. [12]

Levinud JAX-RS implementatsioon on ka Jersey. Kahjuks ei saab Jersey 2 aga Guice-iga hästi läbi. Jersey 2 arendajad otsustasid luua ise oma sõltuvuste süstimise raamistiku HK2, mille nad pakkisid Jersey 2 sisse ja selleks, et see Guice-iga tööle saada on vaja kasutada koodi, mis võib uuemate Jersey väljalasetega lihtsalt katki minna [4].

Google API client teek lihtsustab Google API-dega suhtlemist, seal on olemas näiteks API-de aadressid, meetodid päringute jaoks ja ka päringutes kasutatavad klassid andmete serialiseerimiseks ja deserialiseerimiseks [13]. Rakenduses on see kasutuses Google-i abil sisse logimise jaoks.

Facebook4j on teek, mis lihtsustab Facebook API kasutamist Java rakendustes. See on avatud lähtekoodiga ja Maveniga kasutatav [14]. Rakenduses on see kasutuses Facebook-i abil sisse logimise jaoks.

Hashids on teek numbritest tähejadade genereerimiseks ja vastupidi, tähejadadest numbrite genereerimiseks [15], seda on pikemalt kirjeldatud peatükis 3.5.3.

BCrypt on salasõnade räsimise funktsioon, mis põhineb Blowfish šiffril. Räsi arvutamise keerukust saab parameetriga muuta, mis võimaldab arvutite arvutusvõimekuse kasvades seda järjest tõsta. Rakenduses on kasutusel jBCrypt – Bcrypti implementatsioon Javas [16].

Lombok võimaldab vähedada *boilerplate* koodi hulka Javas. Enamus Lomboki funktsionaalsustest on kasutatavad annotatsioonidega. Kõige lihtsam näide on @Getter ja @Setter annotatsioonid, mis asendavad Javas väga tavapäraseid getter ja setter meetodeid [17]. Koodinäites nr. 1 on näha Spotify API-st saadava pildi objekt koos Lomboki annotatsioonidega.

```
package com.panolude.app.entity.spotify;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class Image {

    private int height;
    private int width;
    private String url;

}
```

Kood 1. Lombok annotatsioonide näide.

Nagu näha, on selline klass kergemini loetav. Lombok pakub ka annotatsioone nagu: @ToString klassile toString meetodi genereerimiseks, @EqualsAndHashCode equals ja hashCode meetodite genereerimiseks, mitmeid annotatsioone konstruktorite defineerimiseks ilma neid tegelikult välja kirjutatamata ja muid [17].

3.1.2 Front-end

Front-end rakenduse baastehnoloogiad on Javascript, HTML ja CSS.

NPM on sõltvuste haldamise teek Javascripti jaoks. Rakendused, mis seda kasutavad omavad faili nimega package.json, milles kirjeldatakse vajalikud sõltuvused ja mis laetakse alla repositooriumist [18].

Bower on sõltuvuste haldamise teek front-end Javascript rakenduste jaoks. Erinevalt NPMist loob Bower lameda sõltuvusmudeli, mis tähendab, et igast sõltuvusest saab olla vaid üks versioon kasutusel. Bower kasutab sõltuvuste alla laadimiseks git protokolli [19]. Boweri abil alla laetavate sõltuvuste defineerimiseks on rakenduses fail nimega bower.json.

Grunt on Javascripti tööde jooksutaja, mis võimaldab automatiseerida kõiksuguseid korduvaid tegevusi – koodi kompileerimine, minifitseerimine, testimine, vormistuse kontrollimine jms. Konfigureerimiseks kasutatakse Gruntfile nimelist faili, kuhu saab erinevaid töid kirja panna [20].

Angular on veebirakenduste loomise raamistik. Saab kasutada HTMLi ja seda laiendada oma komponentidega – võib luua oma HTML elemente, mis asendatakse soovitud koodiga. Raamistik võimaldab sünkroniseerida väärtusi vaate ja kontrolleri vahel (HTML ja Javascript). Angular kasutab sõltuvuste süstimist ning selles loodud komponendid on süstitavad. Kõik see tähendab, et Angulari rakendustel saab olla hea struktuur ja kood on hästi hallatav [21].

Angularil on ka versioon 2, mis on 2016. aasta mai seisuga veel beetas [22]. Angular 2 jaoks kirjutati raamistik täielikult ümber, ning kasutusel on TypeScript [23]. Rakenduses põhineb Angulari esimesel versioonil, kuna see on stabiilne ja sellele on loodud palju teeke kasutajate poolt. Angular 2-ga töötamisega töö autoril kogemus puudub, mistõttu oleks selle valimine tõsiselt tõstnud projekti riske.

SASS on keel, millega kirjeldada veebilehe välimust, mis siis töödeldakse tavaliseks CSS-iks. SASS toetab muutujaid, üksteise sisse pandud reegleid (*nested rules*), failide importimist, koodi taaskasutamist läbi *mixin* direktiivide ja palju muud [24]. Rakenduses on see kasutusel, kuna muutujaid kasutades sai kergelt vahetada kasutajaliidese värve, *mixin* direktiivide abil sai vähendada koodi duplitseerimist ja reeglite üksteise sisse panemisega oli kood loetavam.

Materialize on front-end rakenduste raamistik, mis rakendab Google-i Material Design põhimõtteid ning võimaldab luua lehti, mis on mugavalt kasutatavad nii arvutis kui tahvlil ja mobiilis. Materialize pakub erinevaid valmiskomponente, näiteks nuppude, modaalide, vormide ja palju muu jaoks. [25]. Selleks, et mugavamalt Materialize-i

komponente Angulari maailmas kasutada, on projektis kasutatud angular-materialize teeki, mis pakub mitmele komponentidele Angulari ümbriseid [26].

3.2 Arhitektuur

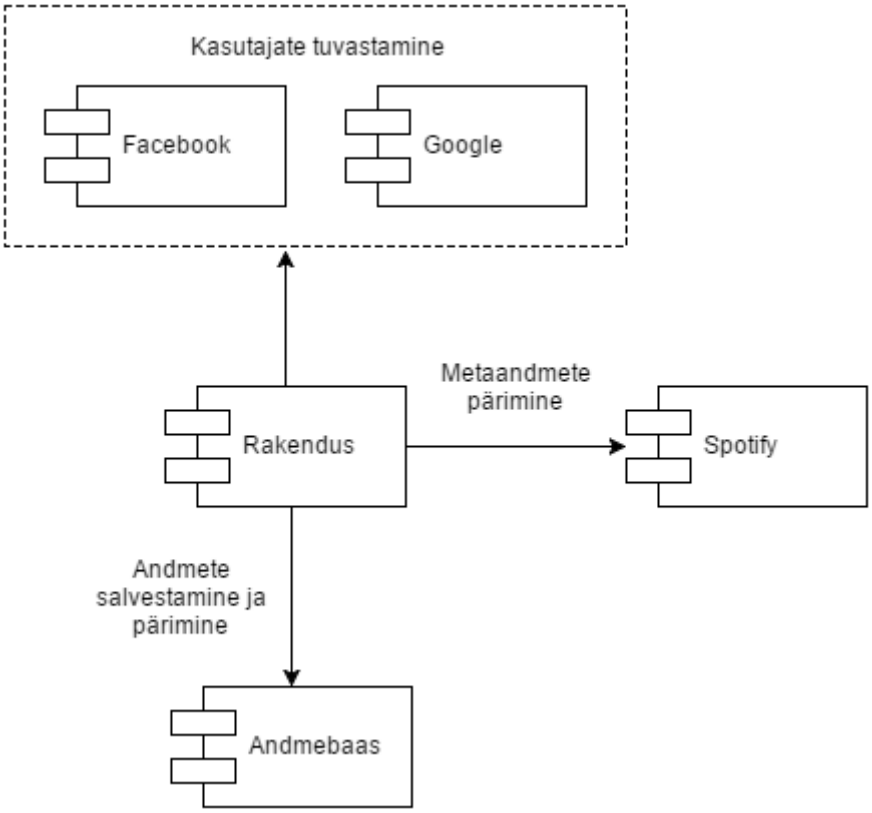
Rakendus on ehitatud kihilise arhitektuuriga. On kolm kihti:

- DAO kiht, mis koosneb klassidest, mille eesmärgiks on andmete püsiv säilitamine ja pärimine. DAO kihi klassid suhtlevad läbi erinevate teekide (käesolevas rakenduses Hibernate-i) andmebaasiga.
- *Service* ehk teenusekiht, mis hoiab endas äriloogikat. Teenusekiht kasutab DAO kihi meetodeid.
- *Resource* ehk ressursikiht, mis suhtleb välismaailmaga, pakkudes REST teenuseid. Ressursikiht kasutab teenusekihi meetodeid.

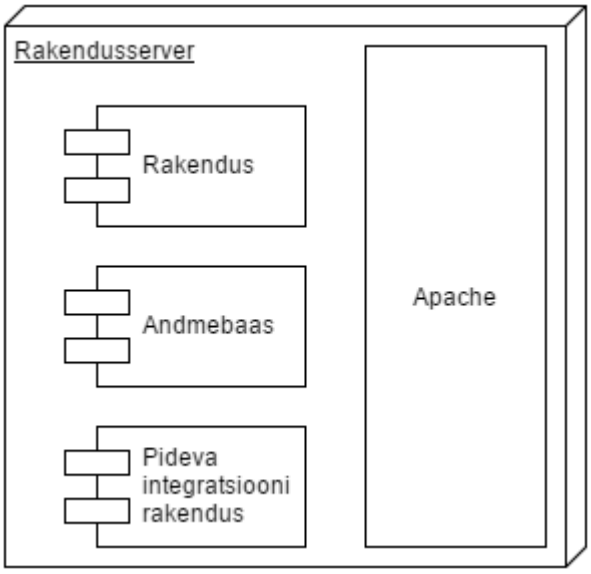
Klasside organiseerimise jaoks on kasutusel *package-by-layer* lähenemine, mis tähendab, et ühte paketti on pandud kõik ühe kihi klassid. Nii on selge, kust igat klassi otsida.

Joonisel nr. 2 on rakenduse lihtne komponentdiagramm. Süsteem koosneb ühest monoliitsest rakendusest. Kuna implementeeritava funktsionaalsuse maht ei olnud väga suur, siis ei olnud rakenduse väiksemateks komponentideks jagamisel otstarvet. Ühe rakenduse puhul on arendamine lihtsam ja kiirem ning päringud saavad kiiremini vastused, kuna ei ole vajalik oodata, et erinevad rakendused omavahel näiteks üle HTTP protokolli suhtleks.

Joonis nr. 3 näitab, et rakendus on paigaldatud samale serverile koos andmebaasi ja pideva integratsiooni rakendusega. Nii on tagatud kiire ja kindel ühendus andmebaasiga ning lihtsus pideva integratsiooni tarkvara seadistamisel. Nii front-end kui back-end rakendus on saadaval läbi Apache. Küll aga teeb back-end ka otse päringuid välistele teenustele.



Joonis 2. Rakenduse komponentdiagramm.



Joonis 3. Paigaldusdiagramm.

3.3 REST liides

Järgnevalt on tabelis nr. 1 toodud ära kõik lubatud päringud REST API-le. Operatsioonide *query parameter* tüüpi parameetrid on toodud ära küsimärgi järel ning *path parameter* tüüpi parameetrid loogeliste sulgude vahel.

Tabel 1. REST liidese ülevaade.

Operatsioon	**Meetod**	**Resurss**
Kasutaja loomine	POST	/register
Sisse logimine	POST	/login
Välja logimine	POST	/logout
Lugude metaandmete päring otsingusõnade järgi	GET	/metadata?query=
Populaarseimate artistide nimekirja küsimine koos metaandmetega	GET	/metadata/mostPopular? maxResults=
Pomodoro loomine	POST	/pomodoros
Pomodorode küsimine aja järgi	GET	/pomodoros?start=&end=
Pomodoro kustutamine	DELETE	/pomodoros/{pomodoroID}
Loo loomine	POST	/songs
Loo uuendamine	PUT	/songs/{songID}
Loo vaatamine	GET	/songs/{songID}
Hetkel sisse logitud kasutaja loodud lugude nimekirja vaatamine	GET	/songs
Loo kustutamine	DELETE	/songs/{songID}
Hetkel sisse logitud kasutaja seadete vaatamine	GET	/settings
Hetkel sisse logitud kasutaja seadete muutmine	POST	/settings

Nendest operatsioonidest ei vaja autentimist vaid registreerimine, populaarseimate artistide nimekirja vaatamine ja loo vaatamine.

3.4 Liidestused teiste süsteemidega

Rakendus liidestub teiste süsteemidega, et pakkuda autentimist ja küsida metaandmeid.

Rakendus kasutab lugude otsimiseks ja metaandmete saamiseks Spotify REST API-t. Kasutusel on kolm erinevat teenust, üks otsimiseks ja kaks täpsemate metaandmete küsimiseks. Järgneb täpsem kirjeldus selle kohta, milliseid teenuseid ja millal kasutatakse.

Lugude otsimiseks kasutatakse teenust https://api.spotify.com/v1/search kus *query parameter* tüüpi parameetritega:

q – antakse kaasa otsingufraas

type – antakse kaasa otsitavate asjade tüüp “track”

limit – antakse kaasa maksimaalne tulemust arv, mida tagastada, hetkel 5.

Päringu vastuseks on JSON formaadis nimekiri lugusid, millel on igal ka lihtsustatud albumi objekt ja lihtsustatud artisti objekt. Lugude nimekiri on omakorda mähitud leheküljestamise (*paging*) objekti [27].

Loo metaandmete küsimiseks kasutatakse teenust https://api.spotify.com/v1/tracks/{id} kus *path parameter* tüüpi parameetriga

id – antakse kaasa loo identifikaator.

Päringu vastuseks on JSON formaadis loo objekt, millel on ka lihtsustatud artisti objekt ja lihtsustatud albumi objekt [28]. Konkreetse loo metaandmeid küsitakse siis, kui kasutaja soovib süsteemi uut lugu lisada ja vastava välise identifikaatoriga – Spotify ID [29] – metaandmeid ei ole varem süsteemi salvestatud.

Artisti metaandmete küsimiseks kasutatakse teenust https://api.spotify.com/v1/artists/{id} kus *path parameter* tüüpi parameetriga

id – antakse kaasa artisti identifikaator.

Päringu vastuseks on JSON formaadis artisti objekt [30]. See päring tehakse siis, kui ühe loo metaandmete küsimisel saadakse lugu, mille artisti täielikke metaandmeid ei ole veel süsteemi salvestatud – nagu varem mainitud, siis /v1/tracks/{id} päring tagastab vaid lihtsustatud artisti objekti, millel puuduvad näiteks viited piltidele. Pildid on aga vajalikud, et kuvada süsteemi esilehel kõige populaarsemaid artiste.

Google abil sisse logimiseks on front-end rakenduses kasutuses google-plus-directive teek [31]. Google tagastab tunnuse (*token*), mis saadetakse back-end-i. Seal tehakse Google Client API [32] teegiga päring, et verifitseerida tunnus ja saada kasutaja info.

Facebook-i abil sisse logimiseks on samuti front-endis teek (angular-easyfb [33]), et teha sisse logimine ja saada *token*. See saadetakse back-end-i, kus facebook4j teegiga [14] tehakse päring kasutaja info jaoks.

3.5 Tehniliste probleemide lahendamine

3.5.1 Transaktsioonide haldus

Üheks Guice mooduliks, mida rakendus kasutab, on RestModule, mis laiendab GWizardi oma RestModule-it ning registreerib filtri (ApplicationPersistFilter), mida käivitatakse iga päringu puhul. Mooduli kood on toodud koodinäites nr. 2.

```
package com.panolude.app.guice.module;

import com.google.inject.Provides;

import com.panolude.app.config.ApplicationConfig;
import com.panolude.app.config.ApplicationWebConfig;
import com.panolude.app.resource.filter.ApplicationPersistFilter;

public class RestModule extends org.gwizard.rest.RestModule {

    @Override
    protected void configureServlets() {
        super.configureServlets();
        filter("/*").through(ApplicationPersistFilter.class);
    }

    @Provides
    public ApplicationWebConfig applicationWebConfig(ApplicationConfig cfg) {
        return cfg.getWeb();
    }

}
```

Kood 2. RestModule koodinäide.

Filtri ülesandeks on alustada ja lõpetada UnitOfWork ehk töö ühik. UnitOfWork tuleneb Guice Persist teegist ning selle avamine tähendab sisuliselt ühenduse loomist andmete kätte saamiseks [34].

Lisaks sellele on JAX-RS ressursiklassides meetoditele määratud @Transactional annotatsioon, mis on samuti Guice Persist teegist. Meetodite jaoks, mis on sellega märgitud, alustatakse uus transaktsioon ja lõpetatakse see siis, kui meetod töö lõpetab. Kui meetod viskab vea, siis veeretatakse transaktsioon tagasi [35].

3.5.2 Autentimine rakenduse kasutamisel

Süsteemi on võimalik sisse logida kasutajanime ja parooliga, Google ja Facebook kontodega. Autentimiseks saadetakse päring POST /login koos kasutajanime ja parooliga või Google/Facebook tokeniga JSON kujul, nagu on näha koodinäites nr. 3.

```
{
    "username": "john",
    "password": "test",
    "googleToken": null,
    "facebookToken": null
}
```

Kood 3. Autentimispäringu näide.

Eduka autentimise vastuseks saadetakse Authentication objekt, mille näide on toodud koodinäites nr. 4. Authentication objekti sees on token ja ka User objekt kasutajanimega.

```
{
    "id" : "NMKbyRQ2",
    "token" : "6rv3r8fnj042ig3vj01vve2ld",
    "user" : {
        "id" : "pVEvLRWy",
        "username" : "john",
        "role" : "USER"
    }
}
```

Kood 4. Autentimispäringu vastuse näide.

Edasiste päringute puhul lisab rakenduse front-end igale päringule HTTP päisesse kaks rida: Username – sisse logitud kasutaja nimi ja Auth-Token – autentimispäringu vastusest saadud token.

OWASPi soovituse järgi peaks sessiooni identifikaator olema vähemalt 128 bitti, et tõkestada *brute-force* rünnakuid. Kui kasutada 128 bitiseid identifikaatoreid ja eeldada, et ründaja suudab teha 10 000 arvamist sekundis ning eksisteerib 100 000 kehtivat identifikaatorit, siis tõenäoliselt kuluks kehtiva identifikaatori leidmiseks üle 292 aasta [36]. Seetõttu valitakse uue tokeni loomiseks krüptograafiliselt turvalisest suvalise

numbri generaatorist 130 bitti ja kodeeritakse need baas 32-te. Selle koodi näide on toodud lisas nr. 1.

3.5.3 Identifikaatorite peitmine

Ressursside identifikaatorite esitamiseks on mitmeid võimalusi. Esimene ja kõige ilmsem, on kasutada objektide reaalseid numbrilisi identifikaatoreid nagu 1, 2, 3 jne. Siis oleks mõne objekti URL näiteks https://example.com/song/12. Teine võimalus on kasutada identifikaatoritena tähtede ja numbrite kombinatsiooni, nagu näiteks YouTube. Siis võiks URL olla selline: https://www.youtube.com/watch?v=dQw4w9WgXcQ.

Selles rakenduses otsustas autor teise variandi kasuks, kuid väikeste mööndustega. Süsteemi siseselt on lihtsam kasutada numbrilisi, järjestikuseid identifikaatoreid. Kasutajale kuvatakse aga tähtede ja numbrite jada, millest ei ole otseselt arusaadav, millisele numbrile see tegelikult vastab.

Objektide reaalsete identifikaatorite mitte avaldamisel on mitu tagajärge:

- Ei ole võimalik mõne ressursi identikaatorit suurendada ja nii kõiki võimalikke ressursse loetleda – ei saa käia läbi aadresse example.com/thing/1, example.com /thing/2, example.com/thing/3 jne.
- Ei ole võimalik teada, mitmenda järjekorranumbriga on konkreetne ressurss. Lihtsa näite võib tuua sellest, et igal süsteemil on kolmas kasutaja, aga ta ei pea seda teadma.
- Ei ole võimalik teada, kui palju on vahepeal ressursse juurde tulnud.

Süsteemis lahendati see kasutades Hashids teeki. Selle kasutatamiseks tuleb valida salajane kood ning seejärel saab konverteerida numbrilisi identifikaatoreid märkide jadaks ja vastupidi – märke numbriks. On võimalik ka seada minimaalne pikkus väljundile [15].

Song objekti lihtsustatud klass on toodud koodinäites nr. 5. Song objekti identifitseerib Long tüüpi väli nimega id. Sellele on rakendatud Jacksoni annotatsioonid, mis kehtestavad sellele spetsiaalse JSON serialiseerija ja deserialiseerija. See tähendab, et iga kord, kui kliendile saadetakse Song objekt, serialiseeritakse id väli kasutades IdentifierSerializer klassi, nii et näiteks identifikaator 1 muutub jadaks “pVEvLRWy”.

Iga kord, kui klient teeb päringu /songs/pVEvLRWy või uuendab loo objekti, konverteeritakse “pVEvLRWy” jälle tagasi numbriks.

```
package com.panolude.app.entity;

import javax.persistence.Entity;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;

import lombok.Getter;
import lombok.Setter;

import com.panolude.app.resource.jackson.IdentiferDeserializer;
import com.panolude.app.resource.jackson.IdentiferSerializer;

@Entity
@Getter
@Setter
public class Song {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonSerialize(using = IdentiferSerializer.class)
    @JsonDeserialize(using = IdentiferDeserializer.class)
    private Long id;

    // other fields omitted

}
```

Kood 5. Identifikaatorite töötlemise näide lihtsustatud Song objekti põhjal.

3.5.4 Konfiguratsioon

Back-end projekti seadistamine käib läbi YAML formaadis faili, mille asukoht tuleb anda kaasa käsureaparameetrina rakenduse käivitamisel, näiteks niimoodi:

java -jar panolude.jar properties.yml

kus panolude.jar on rakendus ja properties.yml on konfigratsioonifail. Konfiguratsioonifail loetakse Jacksoni abil Java objektidesse, rakenduse põhiline konfiguratisooniklass on toodud koodinäites nr. 6.

```
package com.panolude.app.config;

import ...

@Getter
@Setter
public class ApplicationConfig {
	private int shutdownPort;
	private LoggingConfig logging = new LoggingConfig();
	private ApplicationWebConfig web = new ApplicationWebConfig();
	private DatabaseConfig database = new DatabaseConfig();
	private HashidsConfig hashids = new HashidsConfig();
	private AuthenticationConfig auth = new AuthenticationConfig();
}
```

Kood 6. Back-end rakenduse konfiguratsiooniklass.

Kasutatakse Guice providereid, et teha nii põhiline konfiguratsiooniobjekt kui ka selle sees olevad objektid eraldi koodi süstitavaks. Näiteks registreerimise komponent võib küsida Guice-i käest AuthenticationConfig objekti, et saada teada minimaalset lubatud parooli pikkust.

Front-end rakenduses on eraldi javascripti fail, mis tekitab window objektile uue konfiguratsiooniobjekti ja sellele konfiguratsioonimuutujad. Angulari rakenduse käivitamisel loetakse konfiguratsiooniobjekt window objektilt ja salvestatakse Angulari konstandina, nii et seda on võimalik sõltuvusena süstida kontrolleritesse ja teenustesse. Nii saab kergesti muuta seadeid näiteks rakenduse *live* versiooni jaoks kirjutades paigaldamisel vaid ühe faili üle.

3.5.5 Animatsioonid

Harjutusvaates kuvatakse lugude nimekiri ja harjutussessioonid kastikestena vastava ajaperioodi jooksul, selle ekraanitõmmis on lisas nr. 2. Liikudes eelmisele või järgmisele ajaperioodile, toimub hierarhilise ajastuse animatsioon, nagu on kirjeldatud Google Material Design spetsifikatsioonis [37]. Esimesena ilmuvad kastikesed üleval ja vasakul nurgas, viimasena all paremal nurgas.

Loo nime järgi otsimisel on olemas progressiriba, et kasutaja saaks aru, et süsteem on töötamas ja ei vajutaks kohe lihtsalt lisamise nupule. Loo lisamine on samuti animeeritud, uue loo kast tekib sujuvalt tühjusest.

3.6 Testimine

Automaattestid toovad projekti mitmeid häid asju:

- kaitse regressioonide eest – kui uus funktsionaalsus lõhub vana on see kohe teada
- julgus refaktoreerida, sest koodi ikka õigesti töötamist saab kergelt kontrollida [38]
- kood, mis on testitav, on tavaliselt parema ülesehitusega kui selline kood, mis ei ole testitav – koodile testide kirjutamine aitab parandada koodi kvaliteeti

Koodi kattuvuse mõõtmiseks on kasutusel Cobertura. Kattuvust mõõdab Jenkins iga uue versiooni ehitamise ajal. Joonisel nr. 4 on back-end projekti koodi testidega kaetus. Näiteks ridade kaetus oli 87% ja konditsionaalide kaetus 75%.

Project Coverage summary

Name	Packages	Files	Classes	Methods	Lines	Conditionals
Cobertura Coverage Report	94% 17/18	97% 72/74	96% 74/77	84% 312/371	87% 693/796	75% 100/134

Coverage Breakdown by Package

Name	Files	Classes	Methods	Lines	Conditionals
com.panolude.app	0% 0/1	0% 0/1	0% 0/2	0% 0/9	0% 0/2
com.panolude.app.config	100% 3/3	100% 3/3	94% 15/16	94% 16/17	N/A
com.panolude.app.config.auth	100% 3/3	100% 3/3	100% 12/12	100% 12/12	N/A
com.panolude.app.dao	100% 8/8	100% 8/8	100% 26/26	100% 71/71	N/A
com.panolude.app.entity	100% 9/9	100% 9/9	100% 55/55	100% 57/57	N/A
com.panolude.app.entity.auth	100% 2/2	100% 2/2	100% 10/10	100% 10/10	N/A
com.panolude.app.entity.spotify	100% 5/5	100% 5/5	44% 24/55	44% 24/55	N/A
com.panolude.app.entity.spotify.search	100% 2/2	100% 2/2	50% 6/12	50% 6/12	N/A
com.panolude.app.guice	100% 1/1	100% 1/1	67% 2/3	75% 3/4	N/A
com.panolude.app.guice.module	67% 2/3	67% 2/3	87% 13/15	83% 24/29	100% 2/2
com.panolude.app.guice.provider	100% 4/4	100% 4/4	93% 13/14	93% 38/41	100% 6/6
com.panolude.app.resource	100% 10/10	100% 10/10	91% 39/43	93% 70/75	73% 16/22
com.panolude.app.resource.exception	100% 1/1	100% 1/1	100% 3/3	86% 6/7	50% 1/2
com.panolude.app.resource.filter	100% 4/4	100% 4/4	81% 13/16	86% 37/43	63% 5/8
com.panolude.app.resource.jackson	100% 2/2	100% 2/2	100% 6/6	100% 7/7	N/A
com.panolude.app.services	100% 12/12	92% 12/13	94% 62/66	91% 258/285	76% 58/76
com.panolude.app.util	100% 3/3	100% 3/3	63% 5/8	81% 26/32	80% 8/10
com.panolude.app.web	100% 1/1	100% 3/3	89% 8/9	93% 28/30	67% 4/6

Joonis 4. Koodi testidega kaetus.

3.6.1 Unit testid

Testimise jaoks on kasutusel JUnit – raamistik testide kirjutamiseks ja jooksutamiseks [39]. Selleks, et testida DAO klasse, on testkonfiguratsioonis seatud andmebaasiks H2 mälus asuv (*in-memory*) andmebaas ja testandmed loetakse testide alguses import.sql nimelisest failist andmebaasi. Nii saab testida DAO klasse päris andmebaasi vastu ning testid on lühikesed ja kergesti loetavad. Üks test on toodud koodinäites nr. 7.

```
package com.panolude.app.dao;

import static org.junit.Assert.assertEquals;
import java.util.List;
import org.junit.Test;
import com.panolude.app.entity.Artist;
import com.panolude.app.test.DAOTestBase;

public class ArtistDAOTest extends DAOTestBase {

    @Test
    public void findByExternalID() {
        ArtistDAO artistDAO = instance(ArtistDAO.class);
        Artist artist = artistDAO.findByExternalID("asd-artist-asd");

        assertEquals(Long.valueOf(1), artist.getId());
        assertEquals("Young the Giant", artist.getName());
        assertEquals("https://example.com/img/2",
            artist.getImages().get(0).getUrl());
    }

    ...

}
```

Kood 7. DAO klassi testimine.

Koodinäites nr. 7 toodud testmeetodis testitakse, kas rakendus suudab leida identifikaatori järgi Artist objekti. See objekt on testide alguses import.sql failist andmebaasi laetud ja peaks olema selles testis leitav.

Selleks, et testida teenusklasse (*service)* on kasutusel lisaks veel EasyMock – testimise teek, mis aitab isoleerida testitavat klassi muudest klassidest. EasyMock lubab luua mock objekte, mis saavad asendada päris objekte. Nende käitumist saab simuleerida ja nende kasutamist kontrollida [40]. EasyMock töötab väga kenasti koos Guice-iga – kõik sõltuvused, mis muidu süstitaks testitavasse klassi, suudab EasyMock asendada soovitud mock objektidega.

Järgnevalt on koodinäites nr. 8 ühe teenuse test, mis kontrollib, et olemasolevate metaandmete puhul tagastatakse lihtsalt need, mitte ei tehta uut päringut välisele teenusele. @TestSubject annotatsiooniga on märgitud testitav klass. Kõik @Mock annotatsiooniga objektid on need, mis asendavad testitavas klassis olevaid @Inject annotatsiooniga objekte, nende tegevus on simuleeritud ja kontrollitav.

```
package com.panolude.app.services;

import static org.easymock.EasyMock.expect;
import static org.easymock.EasyMock.replay;
import static org.easymock.EasyMock.verify;
import static org.junit.Assert.assertEquals;

import javax.ws.rs.client.Client;

import org.easymock.EasyMockRunner;
import org.easymock.Mock;
import org.easymock.TestSubject;
import org.junit.Test;
import org.junit.runner.RunWith;

import ...

@RunWith(EasyMockRunner.class)
public class SpotifyServiceTest {

    @TestSubject
    private SpotifyService spotifyService = new SpotifyService();

    @Mock private Client client;
    @Mock private MetadataDAO metadataDAO;
    @Mock private ExternalImageDAO externalImageDAO;
    @Mock private ArtistDAO artistDAO;

    @Test
    public void testGetExistingTrack() {
        String id = "we-have-this";
        Metadata existingMetadata = new Metadata();
        expect(metadataDAO.findByExternalID(id)).andReturn(existingMetadata);

        replay(client, metadataDAO, externalImageDAO, artistDAO);

        Metadata metadata = spotifyService.getTrack(id);

        verify(client, metadataDAO, externalImageDAO, artistDAO);

        assertEquals(existingMetadata, metadata);
    }

    ...
}
```

Kood 8. Testimine mock objektidega.

3.6.2 Integratsioonitestid

Integratsioonitestide jaoks käivitatakse kogu rakendus – ka REST back-end ja tehakse sellele päringuid. Nii testitakse kogu rakenduse kõigi kihtide töötamist korraga.

Head testid on sellised, mille käivitamiseks ei ole vaja internetiühendust – st testid ei peaks sõltuma välistest osapooltest. Kuna rakendus kasutab näiteks metaandmete saamiseks välist teenust, siis on sellega seotud funktsionaalsuste testimiseks abiks Guice-i Provider klassid [41]. Nimelt süstitakse klassi tavaliselt Guice poolt tavaline Client objekt, millega saab teha HTTP päringuid. Kasutades Provider klassi on aga võimalik teha nii, et testides süstitaks sinna hoopis selline Client objekt, mis käitub täpselt nii nagu testis vaja. Tulemuseks on test, mis ei sõltu välisest teenusest.

3.7 Pidev integratsioon

Pidev integratsioon ehk Continuous Integration on tarkvara arendamise viis, kus arendajad laevad pidevalt oma koodi jagatud repositooriumisse, mida automaatselt ehitatakse ja kontrollitakse, et juba varakult probleeme leida - toimub nö integreerimine. Tänu sellele on olemas kindlus, et kood töötab. CI server võib jooksutada mahukaid teste, mis muidu koormaks arendajate masinaid. CI serveriga saab automatiseerida ka tarkvara paigaldamise testserverisse [42].

Arendusprotsessi alguses sai serverisse üles seatud CI tarkvara Jenkins. Jenkins konfigureeriti automaatselt kontrollima projekti koodi repositooriume iga 5 minuti tagant. Leides, et vahepeal on back-end projekti koodi uuendatud, toimuvad järgmised tegevused:

- Laetakse alla koodi muudatused.
- Käivitatakse Maven, et kompileerida kood, jooksutada automaatseid teste ja luua artefakt – käivitatav jar fail, mis sisaldab kogu programmikoodi ja ka sõltuvusi.
- Peatatakse vana rakendus
- Kustutatakse vana artifakt, kopeeritakse asemele uus artefakt
- Uuendatakse andmebaasi
- Käivitatakse uus rakenduse versioon

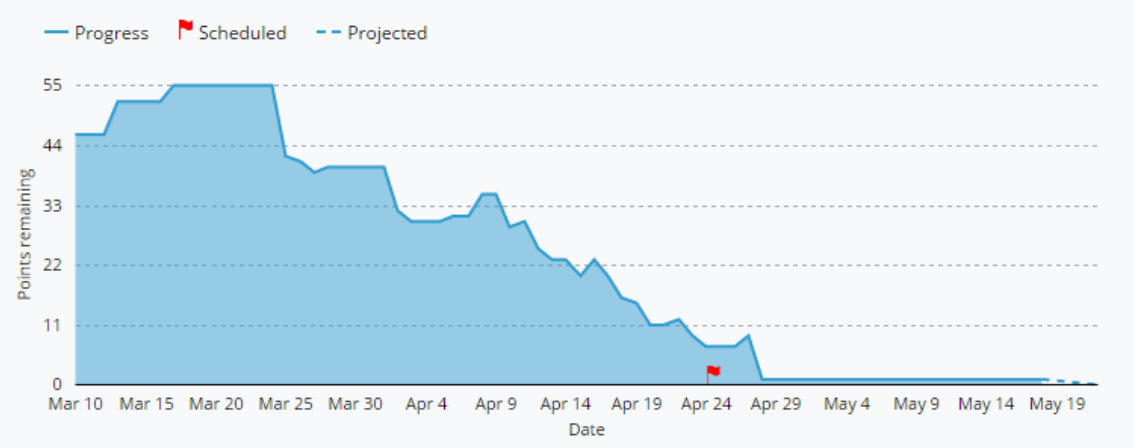
Front-end projekti konfiguratsioon on sarnane, Maveni käskude asemel aga laetakse alla NPM-i ja Boweri abil projekti sõltuvused ning ehitatakse koodist minifitseeritud ja kokku pakitud versioon.

Selles projektis oli suurim kasu Jenkisist neist kahes asjas:

1. Jenkins jooksutas teste üle Maveniga, mis aitas püüda vigu, mis ei tulnud välja IDE-ga teste jooksutades – näiteks võis mõni viga ilmneda olenevalt testide järjekorrast.
2. Jenkins paigaldas automaatselt testserverile uusima rakenduse versiooni, hoides kokku palju aega.

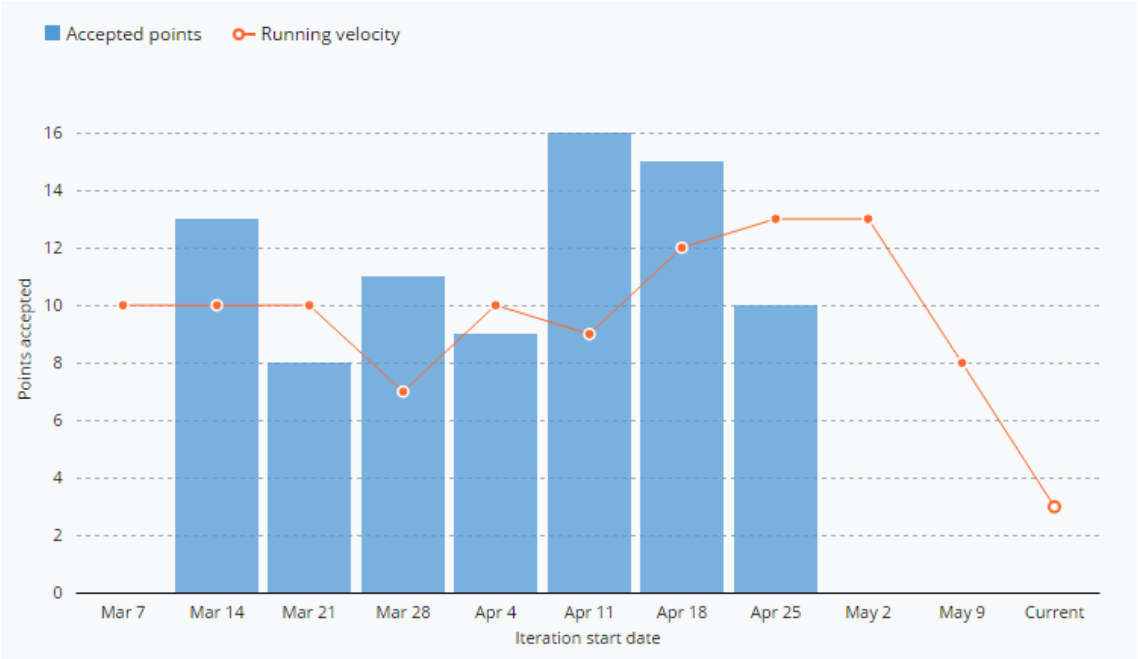
4 Hinnang tulemusele

Projektihalduskeskkonnas Pivotal loodud kasutajalugudele sai määratud suurushinnanguks 1 punkt. Kasutajalugude tegemise ülevaadet on näha joonisel nr. 5, kus horisontaalsel teljel on aeg ja vertikaalsel teljel hetkel veel jäänud punktide arv. Kui punktide arv läheb üles, tähendab see, et sel hetkel lisati juurde kasutajalugusid ja punktide arvu langemine tähendab, et osad said tehtud (või mõni ka plaanist maha võetud ehk *icebox*-i pandud). Arenduse plaanitud lõpuks oli 24. aprill, tegelikult said peaaaegu kõik tööd tehtud 28.aprilliks ning jäi vaid üks kasutajalugu.



Joonis 5. Projekti punktide jäägi progress.

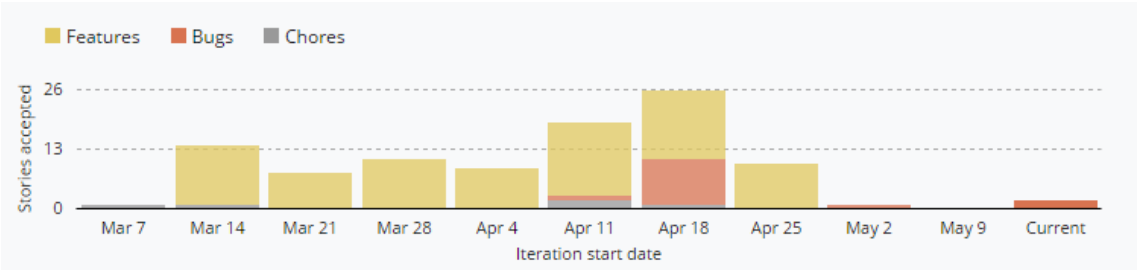
Plaanitud töödest sai arenduse käigus plaanist maha võetud soovituste funktsionaalsus. Vaadates keskmist arenduskiirust, mis oli projekti alguses umbes 11 punkti nädalas (joonis nr. 6), oli näha, et arendus läheb niigi ligikaudu nädala üle planeeritud tähtaja ning selle lisamine ei oleks võimalik. Arenduse käigus tuli ka mõte võimaldada failide lisamist lugudele, kuid ka see sai plaanist maha võetud, sest oli ilmne, et selle arendamine võtaks liialt aega.



Joonis 6. Punktide jagunemine iteratsioonidesse.

Kõik ülejäänud funktsionaalsused peale soovituste, mis on kirjeldatud funktsionaalsetes nõuetes, on arendatud ja töötavad. Valminud veebirakenduse *live* versioon on kasutatav aadressil https://www.panolude.com.

Arenduse käigus tekkis ka vigu, mida tuli parandada. Enamus neist olid seotud kasutajaliidesega, mõned ka näiteks metaandmete liigse duplitseerimisega andmebaasis. Joonisel nr. 7 on iteratsioonidesse kuuluvate kasutajalugude ülesehitus tüübi järgi – kollased on funktsionaalsused, punased on vead ja hallid haldustööd.



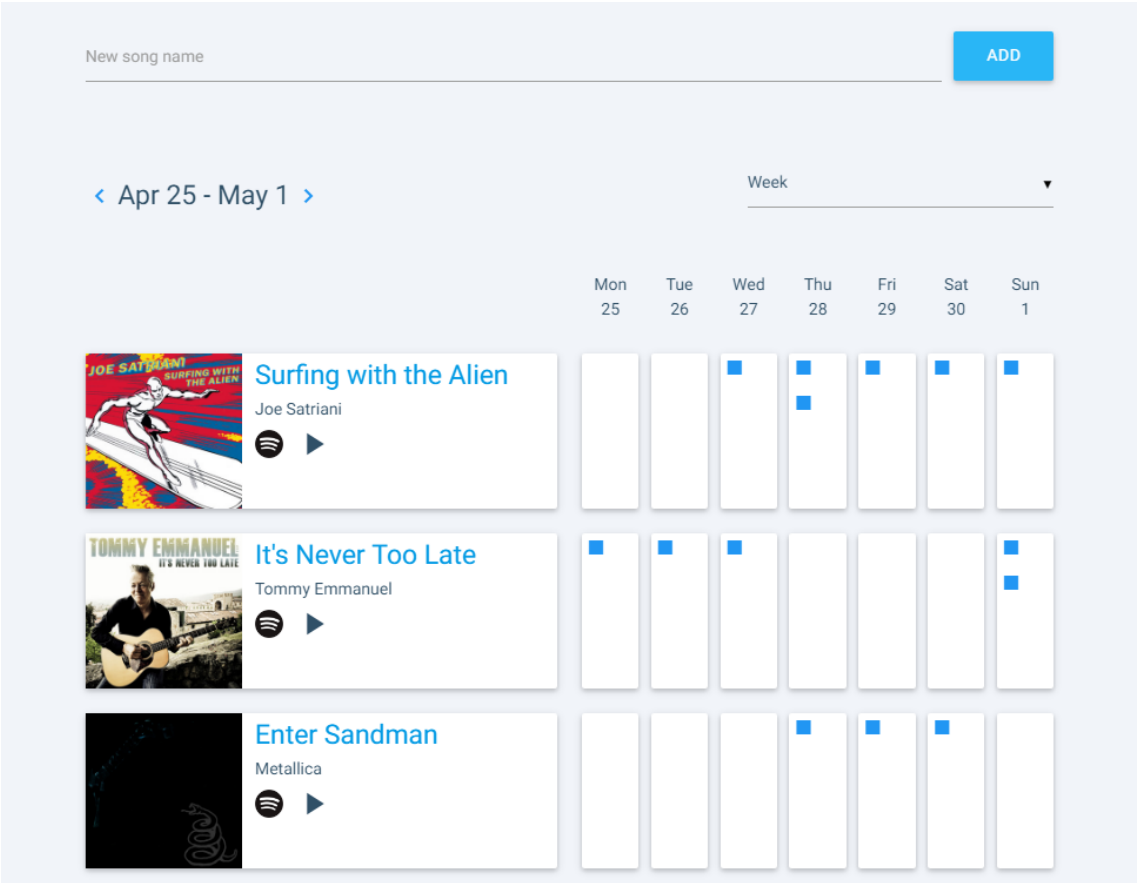
Joonis 7. Kasutajalugude tüüpide jaotus.

4.1 Mobiilivaadete analüüs

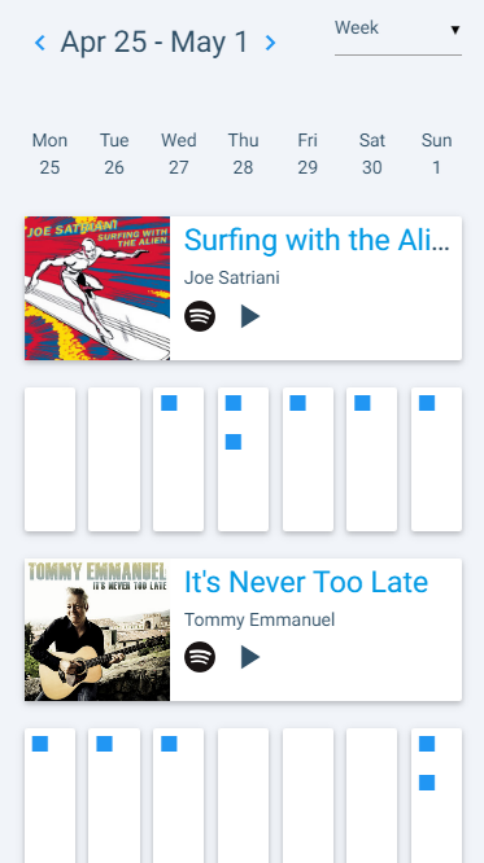
Selleks, et rakendus oleks kasutatav ka mobiiltelefonides, on kasutajaliides ehitatud nii, et elementide paigutus muutuks vastavalt ekraani laiusele. Liidesel on kolm põhilist olekut:

- Lauaarvuti vaade, kui ekraani laius on üle 992 piksli
- Tahvelarvuti vaade, kui ekraani laius on üle 600 ja väiksem või võrdne 992 piksliga
- Mobiilivaade, kui ekraani laius on väiksem võrdne 600 piksliga.

Mobiilivaates paigutatakse suuremad elemendid üksteise alla, tahvelarvuti vaates paigutatakse elemendid mõneti horisontaalselt üksteise kõrvale ning lauaarvuti vaates on veelgi rohkem elemente horisontaalselt üksteise kõrval. Hea näide sellest on harjutusvaade. Joonisel nr. 8 on näha harjutusvaade tahvelarvutil ja joonisel nr. 9 mobiilis. Lauaarvuti vaade on töö lisas number 2.

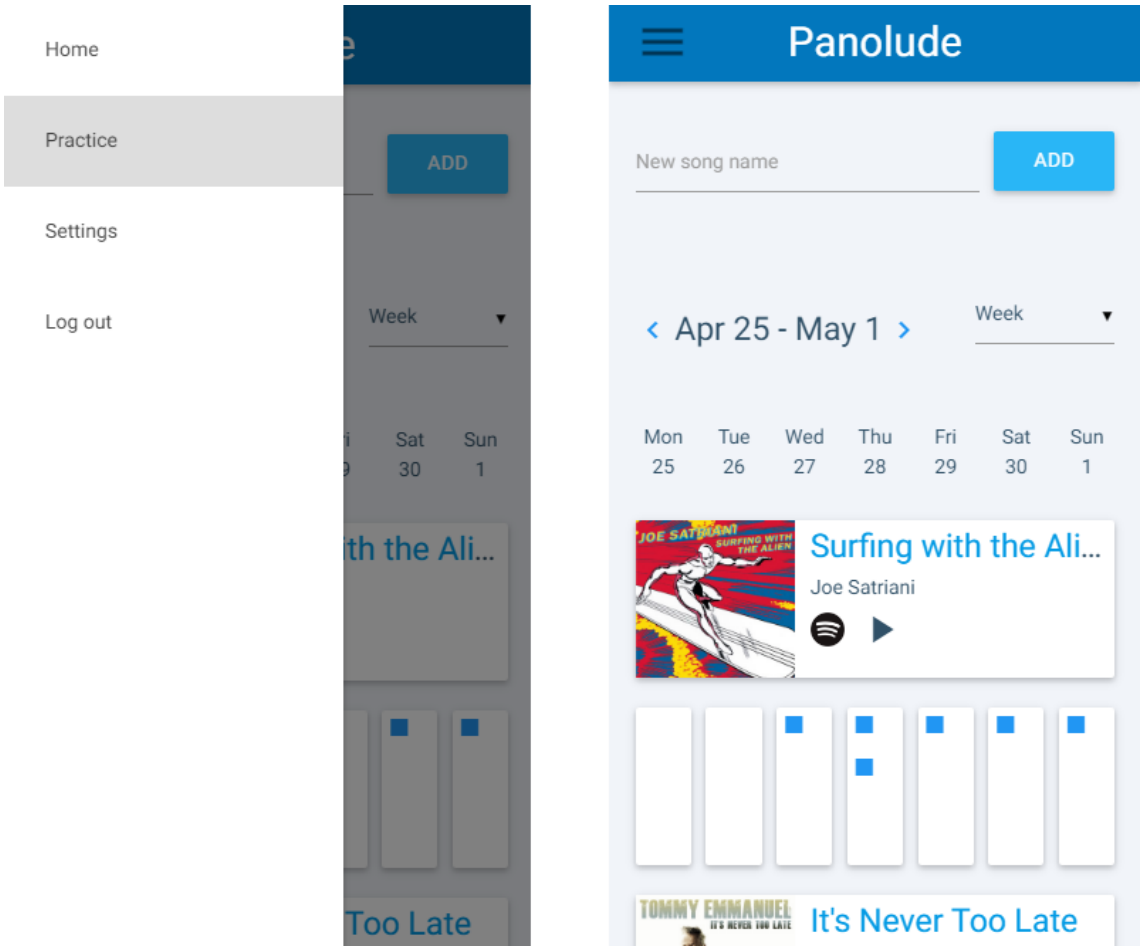


Joonis 8. Harjutusvaade tahvelarvutil.



Joonis 9. Harjtusvaade mobiilivaates.

Tavaliselt on navigeerimislingid sinisel menüüribal paremas ääres. Mobiilis navigeerimiseks on aga vasakul üleval nurgas nupp, millele vajutades avaneb navigatsioonimenüü vertikaalse nimekirjana, seda kirjeldab ka joonis nr. 10.



Joonis 10. Navigatsioon mobiilivaates.

Ülejäänud mobiilivaated on lisas nr. 3.

4.2 Probleemid

Tõenäoliselt vajaks veebirakendus selgitusi selle kohta, mis on Pomodoro tehnika ja kuidas rakendust kasutada. Sellised tekstid võiksid olla kas avalehel, kus praegu sisse logituna ei ole midagi väga kasulikku, või implementeeritud interaktiivse sissejuhatusena, mis sammhaaval juhiks tähelepanu erinevatele liidese elementidele ja õpetaks neid kasutama.

Laulu detailvaates on võimalik lisada linke YouTube videotele. Lisades mitu linki, on võimalik lingi kõrval olevale ikoonile vajutades valida, millist videot lehel hetkel vaadata. Kasutajal on aga seda funktsionaalsust raske ise avastada.

Arenduse käigus oleks võinud ka front-end rakendusele teste kirjutada. Kuna aga autoril töö alustades sellega kogemus puudus ja aeg oli väga piiratud, siis seda ei tehtud.

5 Kokkuvõte

Töö põhieesmärk oli luua rakendus, et aidata üles märkida märkida muusikainstrumendi harjutamist ja saada sellest paremaid ülevaateid kui lihtsalt paberile märkmeid tehes. Eesmärgiks oli ka siduda rakendus Pomodoro tehnikaga.

Töö jagamine kasutajalugudeks aitas aru saada, kui palju tööd oli veel teha. Tarkvara testimine andis kindluse, et arenduse käigus loodud funktsionaalsused töötavad. Töö lugeja peaks saama siit ülevaate rakenduse tehnoloogilisest ja arhitektuurilisest ülesehitusest ning mõningate tehnoloogiliste probleemide lahendamisest.

Peaaegu kogu planeeritud funktsionaalsus sai loodud. Töö tulemuseks on reaalne ja kasutatav tarkvara, mida on võimalik kergesti edasi edasi arendada ja uute funktsionaalsustega täiendada.

6 Kasutatud kirjandus

[1] „Maven projekti koduleht. What is Maven?," [Võrgumaterjal]. Available: https://maven.apache.org/what-is-maven.html. [Kasutatud 30 4 2016].

[2] „Maven projekti koduleht. POM," [Võrgumaterjal]. Available: https://maven.apache.org/pom.html. [Kasutatud 30 4 2016].

[3] „Maven Central Repository," [Võrgumaterjal]. Available: https://search.maven.org/. [Kasutatud 30 4 2016].

[4] „GWizard GitHub projekti readme," [Võrgumaterjal]. Available: https://github.com/gwizard/gwizard/blob/master/README.md. [Kasutatud 30 4 2016].

[5] „Guice GitHub projekti readme," [Võrgumaterjal]. Available: https://github.com/google/guice/blob/master/README.md. [Kasutatud 30 4 2016].

[6] „Guice projekti dokumentatsioon. Getting Started," [Võrgumaterjal]. Available: https://github.com/google/guice/wiki/GettingStarted. [Kasutatud 30 4 2016].

[7] „Guice projekti dokumentatsioon. Injection," [Võrgumaterjal]. Available: https://github.com/google/guice/wiki/Injections. [Kasutatud 30 4 2016].

[8] „Hibernate ORM," [Võrgumaterjal]. Available: http://hibernate.org/orm/. [Kasutatud 30 4 2016].

[9] „Hibernate ORM 5.1 Quickstart," [Võrgumaterjal]. Available: https://docs.jboss.org/hibernate/orm/5.1/quickstart/html_single/. [Kasutatud 30 4 2016].

[10] „Guice Persist dokumentatsioon," [Võrgumaterjal]. Available: https://github.com/google/guice/wiki/GuicePersist. [Kasutatud 5 5 2016].

[11] „Jackson projekti dokumentatsioon," [Võrgumaterjal]. Available: https://github.com/FasterXML/jackson. [Kasutatud 5 5 2016].

[12] „RESTEasy koduleht," [Võrgumaterjal]. Available: http://resteasy.jboss.org/. [Kasutatud 5 5 2016].

[13] „API Client Library for Java," [Võrgumaterjal]. Available: https://developers.google.com/api-client-library/java/. [Kasutatud 15 5 2016].

[14] „Facebook4J veebileht," [Võrgumaterjal]. Available: http://facebook4j.org/en/index.html. [Kasutatud 15 5 2016].

[15] „Hashids Java teegi Readme GitHubis," [Võrgumaterjal]. Available: https://github.com/jiecao-fm/hashids-java/blob/master/README.md. [Kasutatud 7 5 2016].

[16] „JBCrypti projekti koduleht," [Võrgumaterjal]. Available: http://www.mindrot.org/projects/jBCrypt/. [Kasutatud 15 5 2016].

[17] „Project Lombok koduleht. Lombok features," [Võrgumaterjal]. Available: https://projectlombok.org/features/index.html. [Kasutatud 5 5 2016].

[18] „What is NPM?," [Võrgumaterjal]. Available: https://docs.npmjs.com/getting-

started/what-is-npm. [Kasutatud 15 5 2016].

[19] „Bower GitHub projekti Readme," [Võrgumaterjal]. Available: https://github.com/bower/bower/blob/master/README.md. [Kasutatud 15 5 2016].

[20] „GruntJS," [Võrgumaterjal]. Available: http://gruntjs.com/. [Kasutatud 15 5 2016].

[21] „AngularJS GitHub projekti Readme," [Võrgumaterjal]. Available: https://github.com/angular/angular.js/blob/master/README.md. [Kasutatud 15 5 2016].

[22] „Angular 2 GitHub projekti Readme," [Võrgumaterjal]. Available: https://github.com/angular/angular/blob/master/README.md. [Kasutatud 15 5 2016].

[23] S. Dobrev, „Will Angular 2 be a success? You bet!," [Võrgumaterjal]. Available: http://developer.telerik.com/featured/will-angular-2-be-a-success-you-bet/. [Kasutatud 15 5 2016].

[24] „SASS GitHub projekti Readme," [Võrgumaterjal]. Available: https://github.com/sass/sass/blob/stable/README.md. [Kasutatud 15 5 2016].

[25] „MaterializeCSS," [Võrgumaterjal]. Available: http://materializecss.com/. [Kasutatud 15 5 2016].

[26] „Angular-materialize," [Võrgumaterjal]. Available: https://krescruz.github.io/angular-materialize/. [Kasutatud 15 5 2016].

[27] „Spotify Web API reference. Search item," [Võrgumaterjal]. Available: https://developer.spotify.com/web-api/search-item/. [Kasutatud 1 5 2016].

[28] „Spotify Web API reference. Get track," [Võrgumaterjal]. Available: https://developer.spotify.com/web-api/get-track/. [Kasutatud 1 5 2016].

[29] „Spotify Web API reference. Spotify URIs and IDs," [Võrgumaterjal]. Available: https://developer.spotify.com/web-api/user-guide/#spotify-uris-and-ids. [Kasutatud 1 5 2016].

[30] „Spotify Web API reference. Get artist," [Võrgumaterjal]. Available: https://developer.spotify.com/web-api/get-artist/. [Kasutatud 1 5 2016].

[31] „Angular Google Plus Sign-in Button Directive," [Võrgumaterjal]. Available: https://github.com/sirkitree/angular-directive.g-signin. [Kasutatud 17 5 2015].

[32] „Google Sign-In for Websites: Authenticate with a backend server," [Võrgumaterjal]. Available: https://developers.google.com/identity/sign-in/web/backend-auth. [Kasutatud 17 5 2016].

[33] „angular-easyfb GitHub projekt," [Võrgumaterjal]. Available: https://github.com/pc035860/angular-easyfb. [Kasutatud 17 5 2016].

[34] „Guice Javadoc. Interface UnitOfWork," [Võrgumaterjal]. Available: https://google.github.io/guice/api-docs/latest/javadoc/index.html?com/google/inject/persist/UnitOfWork.html. [Kasutatud 15 5 2016].

[35] „Guice Javadoc. Annotation Type Transactional," [Võrgumaterjal]. Available: https://google.github.io/guice/api-docs/latest/javadoc/index.html?com/google/inject/persist/Transactional.html. [Kasutatud 15 5 2016].

[36] „OWASP: Insufficient Session-ID Length," [Võrgumaterjal]. Available: https://www.owasp.org/index.php/Insufficient_Session-ID_Length. [Kasutatud 5 5

2016].

[37] „Google Material Design. Animation - Meaningful transistions: hierarhical timing,“ [Võrgumaterjal]. Available: http://www.google.com/design/spec/animation/meaningful-transitions.html#meaningful-transitions-hierarchical-timing. [Kasutatud 8 5 2016].

[38] „Unit Tests,“ [Võrgumaterjal]. Available: http://www.extremeprogramming.org/rules/unittests.html. [Kasutatud 17 5 2016].

[39] „JUnit FAQ,“ [Võrgumaterjal]. Available: http://junit.org/junit4/faq.html. [Kasutatud 15 5 2016].

[40] „EasyMock,“ [Võrgumaterjal]. Available: http://easymock.org/. [Kasutatud 15 5 2016].

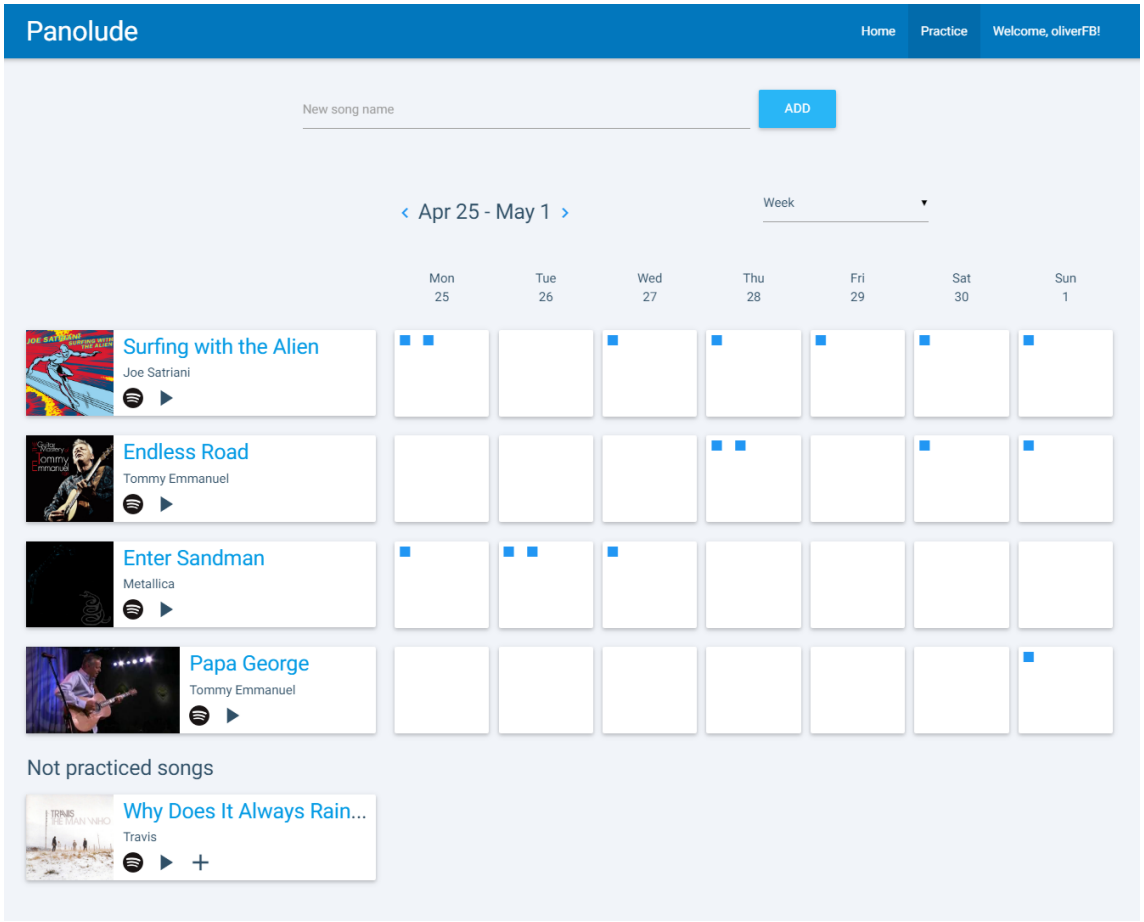
[41] „Google Guice Wiki: @Provides Methods,“ [Võrgumaterjal]. Available: https://github.com/google/guice/wiki/ProvidesMethods. [Kasutatud 15 5 2016].

[42] „Thoughtworks - Continuous Integration,“ [Võrgumaterjal]. Available: https://www.thoughtworks.com/continuous-integration. [Kasutatud 5 5 2016].

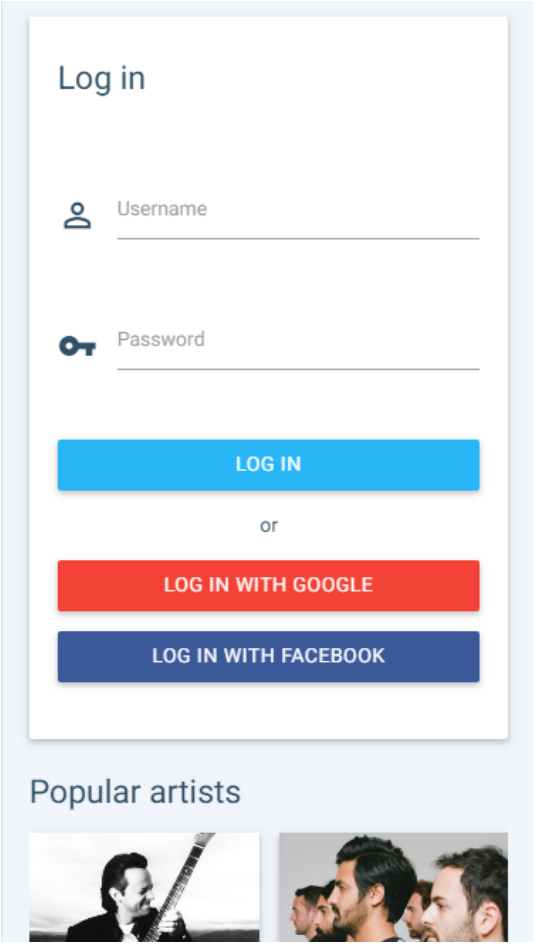
Lisa 1 – Sessiooni ID genereerimise koodinäide

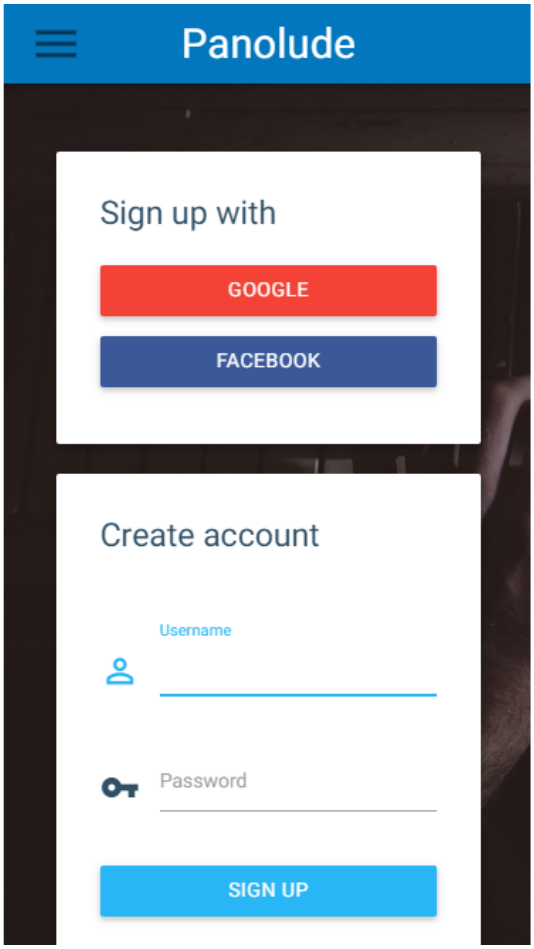
```
package com.panolude.app.services;

import static ...;

import java.math.BigInteger;
import java.security.SecureRandom;

import ...;

@Slf4j
@Singleton
public class LoginService {

    @Inject private UserDAO userDAO;
    @Inject private AuthenticationDAO authenticationDAO;
    @Inject private GoogleService googleService;
    @Inject private FacebookService facebookService;

    private SecureRandom random = new SecureRandom();

    public Authentication login(String username, String password) {
        ...
    }

    public Authentication loginWithGoogle(String token) {
        ...
    }

    public Authentication loginWithFacebook(String token) {
        ...
    }

    private Authentication getAuthentication(User user) {
        Authentication authentication = new Authentication();
        authentication.setUser(user);
        authentication.setToken(new BigInteger(130, random).toString(32));

        return authenticationDAO.update(authentication);
    }
}
```

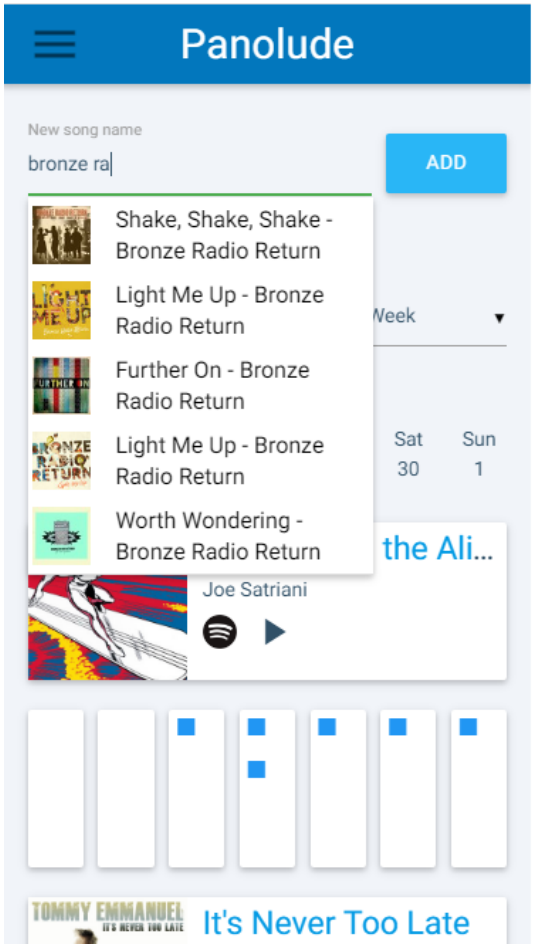
Lisa 2 – Harjutusvaade

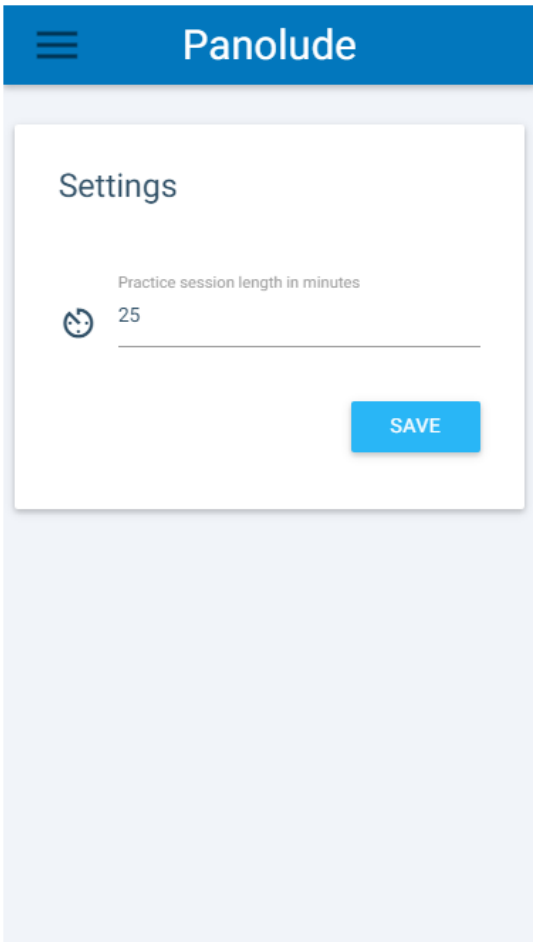


Lisa 3 – Mobiilivaated






Panolude
New song name
bronze ra
ADD
Shake, Shake, Shake - Bronze Radio Return
Light Me Up - Bronze Radio Return
Further On - Bronze Radio Return
Light Me Up - Bronze Radio Return
Worth Wondering - Bronze Radio Return
Week
Sat
30
Sun
1
the Ali...
Joe Satriani
TOMMY EMMANUEL
It's Never Too Late


Panolude
Settings
Practice session length in minutes
25
SAVE