TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Martin Sooväli 179585IADB

PDF-piletite genereerimise süsteemi kaasajastamine ettevõtte Ridango AS näitel

bakalaureusetöö

Juhendaja: Jaanus Pöial
PhD

Tallinn 2021

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: Martin Sooväli

27.04.2021

Annotatsioon

Lõputöös luuakse ettevõttele Ridango AS uus PDF-piletite genereerimise süsteem. Olemasolev süsteem on raskesti arendatav, kuna kasutusel on aegunud tehnoloogiad. Samuti ei ole tehnoloogiaid valides mõeldud, et rakendust peavad arendama peamiselt eesrakendustele orienteeritud arendajad.

Töö analüüsis selgitatakse välja sobiv lahendus, et rakendus võimaldaks eesrakenduse arendajatel hõlpsasti luua uusi piletimalle. Analüüsi tulemuse pealt loob töö autor uue PDF-piletite genereerimise süsteemi võttes arvesse olemasoleva süsteemi probleeme ja puuduseid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 7 peatükki, 20 joonist, 2 tabelit.

Abstract

Modernization of the PDF Tickets Generation System on the Example of Ridango AS

The aim of the thesis is to develop new PDF ticket generation system for Ridango AS. Current system for this purpose is not easy to develop since it uses outdated technologies. Also, when currently in use system was developed then it was not thought through that new ticket templates are created by frontend developers.

Thesis analysis has focus on how to create system that makes use of technologies which are easy to understand for company's frontend developers. After analysis is completed new project will be developed with all the problems and shortcomings taken into account from currently used system.

The thesis is in Estonian and contains 29 pages of text, 7 chapters, 20 figures, 2 tables.

Lühendite ja mõistete sõnastik

Bitbucket pipeline	Bitbucket koodivaramusse kirjutatud skript rakenduse ehitamiseks ja paigaldamiseks
CSS	*Cascading Style Sheets*, stiilikeel millega kirjeldada veebilehe väljanägemist
CSS *grid*	Kahedimensionaalne ruudustiku süsteem võimaldamaks määrata kasutajaliidese paigutust
Docker konteiner	Operatsioonisüsteemi tuuma jagav virtualiseerimiskeskkond
DTO	*Data Transfer Object*, andmete edastamiseks kasutatav objekt
HTML	*HyperText Markup Language*, keel millega määrata veebilehe struktuur
HTTP	*HyperText Transfer Protocol*, võrguprotokoll hüperteksti edastamiseks
IDE	*Integrated Development Environment*, graafiline kasutajaliides kasutamaks programmeerimiseks vajalikke vahendeid
JAR	*Java Archive*, Java koodi pakendamiseks kasutatav vorming
JSON	*JavaScript Object Notation*, andmevahetusvorming
JWT	*JSON Web Token*, identsustõend
Mikroteenus	Ühte kindlat funktsionaalsust täitev rakendus
NPM	*Node Package Manager*, NodeJS paketihaldur
PDF	*Portable Document Format*, tarkvaraplatvormist sõltumatu dokumendi vorming
QR-kood	*Quick Response code*, kahedimensionaalne triipkood, mis võimaldab edastada andmeid
REST	*Representational State Transfer*, tarkvaraarhitektuur, mis seab veebirakendustele kindlad reeglid suhtlemiseks
SPA	*Single Page Application*, üheleherakendus. Serverist laetakse rakendus ühe korra ning edaspidi Javascript töötleb veebilehitsejas kuvatavat sisu
WAR	*Web Application Resource*, Java koodi pakendamiseks ja jooksutamiseks Tomcat serveril kasutatav vorming

Sisukord

1 Sissejuhatus

PDF-vormingus piletid ning arved on saanud kaasaegsete ettevõtete standardseks vahendiks edastamaks kasutajatele andmeid tarbitavate teenuste ja toodete kohta. Võib öelda et PDF-piletite välimus ning kasutusmugavus kajastavad ka ettevõtte käekäiku. Mugava ja korrektse välimusega väljastatud pilet kujundab kasutaja eelistuse teenusepakkuja valikul ning aitab ettevõttel seetõttu ka konkurentsis püsida.

Ridango AS on üks suurimaid ühistranspordi lahenduste ning piletite müügitarkvara arendamisega tegelevaid ettevõtteid Põhja- ning Ida-Euroopas. Ettevõte pakub teenuseid mitmetele ühistranspordi vedajatele, kellel on lahenduste osas erinevad soovid nii äriloogilisel kui ka visuaalsel tasandil.

Käesolevas töös analüüsitakse Ridango AS-i olemasolevat lahendust PDF-piletite väljastamiseks ning sellega kaasnevaid probleeme. Autor loob töö lõpuks ettevõtte struktuurile uue võimalikult optimaalse lahenduse PDF-vormingus piletite genereerimiseks ning erinevatele klientidele piletimallide arendamiseks. Analüüsis selgitatakse välja lõpptulem kaaludes erinevaid tehnoloogiaid ning lahendusi.

Lahendus valmib arvestades ettevõtte Ridango AS nõudeid ning leiab edaspidi kasutust senise mikroteenuse asemel.

1.1 Taust

Töö valmimise hetkel on ettevõttes kasutusel 2 erinevat PDF-piletite genereerimise süsteemi. Kauem kasutusel olnud süsteem baseerub vabavaralisel *JasperReports* teegil, mis on võrdlemisi keerukas nii arendamise kui ka ülalpidamise vaates. Sellest süsteemist töös palju juttu ei tule, kuna üldine eesmärk on äril liikuda edasi dünaamilisema ettevõttesiseselt arendatud süsteemiga.

Teine PDF-piletite genereerimise süsteem on arendatud ettevõttesiseselt aastal 2018 ning kasutab Java programmeerimiskeelt ja veidi iganenud Java teeke. See süsteem on kirjutamise hetkel kasutusel vaid teenusetarbija AS Elroni piletite genereerimisel. Süsteemi eesmärk oli esialgu asendada *JasperReports* süsteemiga genereeritavad PDF-piletid, aga teatud põhjustel ei ole arendatud süsteem olnud kuigi jätkusuutlik.

1.2 Probleem

Probleeme hetkel kasutusel oleva süsteemiga on mitmeid. Esiteks kasutab süsteem *Flyingsaucer* Java teeki, mille abil genereeritakse HTML-st PDF dokument. See teek seab piirangu, mis võimaldab kasutada piletimallide kujunduse määramiseks vaid stiilikeele CSS versiooni 2.1 atribuute, kuigi CSS versiooni 3 osad moodulid on laialdasemalt kasutusel juba aastast 2011 [1]. Selline piirang muudab uute mallide arendamise keeruliseks, kuna eesrakenduse arendaja sooviks kasutada uuemat stiilikeele funktsionaalsust ning atribuute, mis on kasutusel ka ettevõtte veebirakenduste arendamisel.

Teiseks probleemiks on mallide arendamisel kasutusel olev *Velocity template engine* (eestik. mallimootor). Mallimootor asendab piletimallis puuduolevad väljad keskteenuse poolt sisseantavate andmetega. Uusi piletimalle arendatakse võrdlemisi harva ning põhiliselt kasutavad eesrakenduse arendajad igapäevaselt ettevõtte veebide arendamiseks *Angulari* Javascripti raamistikku, mille mallimootori süntaks erineb märgatavalt *Velocity* pakutavast.

Veel üheks probleemiks on CSS stiili kasutamine HTML koodi sees. Nimelt ei võimalda praegune lahendus stiile eraldi stiilifailidena hoida ning nii HTML kui ka stiili atribuudid tuleb paigutada samasse faili. See on üpriski halb lahendus, kuna eesrakendust arendatakse ettevõttes võimalikult taaskasutatavate moodulitena. Kui aga lehtede stiil paikneb samas failis HTML-ga, siis ei ole võimalik luua taaskasutatavaid stiili komponente.

Lisaks on kasutusel olevas rakenduses tarvis mitut erinevat mallifaili, et toetada erinevaid keeli. See ei ole hea lahendus, kuna võib juhtuda, et erinevad failid ei anna enam edasi sama funktsionaalsust ja sisu.

1.3 Eesmärk

PDF-pileti mallide arendus peaks tulevikus olema eesrakenduse arendajatele lihtsam. Samas peaks lahendus olema ka võimalikult dünaamiline, et täita erinevate teenusetarbijate soove, kaasates malli väljaarendamisse lisaks eesrakenduse arendajatele ka veebidisaineri. Ehk teisisõnu ei saa vajaminev lõpptulem olla mugavalt konfigureeritav lõppkasutaja malliarenduse keskkond, vaid eesrakenduse arendajale suunatud võimalikult selge arendusprojekt.

Lõpptulem peab samuti olema jätkusuutlik ning võiks tulevikus vajada võimalikult vähe tehnoloogilisi uuendusi. Hetkel kasutusel olevast rakendusest on näha, et juba arendamise käigus on kasutatud iganenud tehnoloogiaid ning lõpptulem on väga keeruliselt arendatav projekt.

Lõputöö autor on spetsialiseerunud keskteenuste arendamisele ning loodab töö kirjutamise jooksul saada praktilise kogemuse, kuidas toimub ettevõttes eesrakenduste arendamine. Kogemus võimaldab autoril hakata lisaks keskteenusele vajadusel arendama ka eesrakendusi.

1.4 Ülevaade tööst

Töö teine peatükk keskendub olemasoleva süsteemi ja täiendavate nõudmiste kirjeldamisele. Kolmas peatükk analüüsib probleemi ja kaalub erinevaid lahendamise tehnoloogiaid. Analüüsi tulemusena selgub, millised tehnoloogiad probleeme kõige paremini lahendavad. Neljas ja viies peatükk kirjeldavad lahenduseks vajamineva kahe erineva mikroteenuse valmimist. Kuues peatükk keskendub valminud teenuste integreerimisele Ridango süsteemi ning võrdleb valminud lahendust juba kasutusel oleva lahendusega.

2 Nõuded PDF-piletite funktsionaalsusele ja väljanägemisele

Siin peatükis tuuakse välja rakenduse visuaalsed ja funktsionaalsed nõuded. Nõuete kirjeldamiseks kasutatakse olemasolevas rakenduses kasutusel olevat Elron AS-ile arendatud PDF-piletimalli.

2.1 Olemasoleva rakenduse kirjeldus

Nõuete esiletoomiseks tuleb alustada olemasoleva rakenduse kirjeldusest.

2.1.1 Rakenduse põhitehnoloogiad

Elron AS-ile PDF-piletite genereerimiseks on ettevõttes kasutusel eraldi mikroteenus. Teenus on kasutatav vaid läbi keskteenuse, mis kogub andmebaasist kokku PDF-i genereerimiseks vajalikud piletite andmed ning edastab PDF-i genereerimise mikroteenusele. Projekt on arendatud programmeerimiskeeles Java kasutades Spring Boot raamistikku. Esialgu kasutas projekt ehitusvahendina *Maveni*, aga hiljem mindi ettevõttes kõigi Java projektidega üle *Gradle* ehitustööriista peale. Selline muudatus tegi Java rakenduste ehitamise kiiremaks [2] ja ka mugavamaks juba seetõttu, et kõik Java teenused kasutasid ehitamiseks sama vahendit. Rakenduse pakendamiseks oli varasemalt kasutusel WAR-vorming, mis võimaldas rakendust jooksutada *Tomcat* serveril. Hiljem mindi üle JAR-vormingule, mille jooksutamiseks piisas vaid Java käivituskeskkonna olemasolust. Samuti mindi virtuaalmasinate pealt üle *Docker-i* konteinerite peale.

Ettevõte ei sea kasutatavatele tehnoloogiatele piirangut, aga rõhutab, et võimalusel kaaluda variante, mida oskavad ettevõtte arendajad juba hallata.

2.1.2 Rakenduse funktsionaalsuse kirjeldus

Nagu juba mainitud, siis kutsub rakendust välja ettevõtte põhiteenus, mis otsib andmebaasist PDF-i koostamiseks vajalikud andmed kokku. Täpne teenuse toimimise skeem on välja toodud töö lisas 2. Teenus toimib üle REST-arhitektuuri võttes päringuna JSON-struktuuris sisse kõik andmed, mida on vaja PDF-piletil kuvada. Erandiks on QR-kood, mis genereeritakse kaardi numbri põhjal rakenduse siseselt kasutades *Google ZXing Core* teeki. Täpne andmete vastuvõtu struktuur on välja toodud töö lisas 3.

Esialgu valideeritakse rakenduses sissetulevad andmed. Vajadusel tagastatakse põhiteenusele HTTP-staatuskood 400 *Bad Request* koos valideerimisvea kirjeldusega. Edasi vaatab rakendus millist piletimalli ning mis keeles põhiteenus küsis. Selle põhjal valitakse välja sobiv piletimall. Seejärel pannakse vajadusel kokku QR-kood. Lõpuks, kui andmed on olemas, pannakse kokku HTML kasutades *Velocity* mallimootorit. *Velocity* malli näite osa on saadaval lisas 4. Peale HTML-i valmimist kasutab rakendus vabavaralist *Flyingsaucer* teeki, mis genereerib HTML-st PDF-pileti.

2.1.3 Visuaalsed nõuded

Elron AS-i pileti PDF on suurusega A4 ning koosneb suures plaanis kolmest jaotisest, milleks on dokumendi päis, ostukorvi piletite loend koos summaga ning jalus. Dokument algab päisest, kus on lehe vasakule serva joondatud Elroni logo ja ostukorvi või pileti number ning ostukuupäev. Lisaks on Elroni piletitel ka tagastuskood juhuks kui soovitakse ostukorvi pileteid tagastada. Paremale päisesse on joondatud QR-kood, milles sisaldub reisikaardi number. Ostukorvi piletite loendis on üldjuhul joondusega vasakule toodud välja reisi algusaeg ja lõppaeg, rongi liininumber, piletitüüp ja kogus. Esimese klassi piletitel on lisaks toodud ka traadita võrgu parool ja istekoha number. Joondusega paremale on piletite loendis toodud välja reisi kuupäev ning iga pileti all ka pileti maksumus. Vahel on reisidele määratud ka Elroni-poolsed teated, mis kajastuvad samuti piletite loendis. Jalusesse on paigutatud reisijale Elroni-poolne meelespea, mida peab teadma piletit kasutades. Lisaks on jaluses Elroni kontaktinformatsioon. Elroni pileti näide on toodud lisas 5.

2.2 Ootused valmivale rakendusele

Siin peatükis tuuakse välja Ridango AS ootused valmivale rakendusele ning muud tehnilised näitajad.

2.2.1 Ridango AS ootused

Töö teema jõudis autorini kui tekkis vajadus arendada olemasolevat PDF-i genereerimise teenust ka teistele klientidele peale Elron AS-i ning vajadus ka mitme QR-koodi järele ühel PDF-dokumendil. Rakendus oli esialgu arendatud keskteenuse arendaja poolt. Seega ei olnud rakenduse süntaks ja struktuur kuigi lähedased eesrakenduste arendajatele. Eesrakenduse arendajad olid küll varem *Velocity* malle projektis täiendanud, aga sellest oli vahepeal möödunud üle aasta ning arendajad ei mäletanud enam, kuidas toimib *Velocity* mallimootori süntaks ning loogika. Sellest kujunes välja ülesanne uurida, kuidas oleks võimalik uusi PDF-malle kiiremini ja lihtsamalt arendada.

Peale arendamise lihtsuse ja kiiruse tuleb arvestada ka, et erinevatel teenuse tellijatel on väga isemoodi nõuded, mis pileti või arve peal olema peab ning kuidas see välja näeb. Seega tuleb arvestada, et rakendus oleks võimalikult paindlik vastu võtma kujunduse ja funktsionaalsuse muudatusi. Samuti on siiani olnud raskendatud tõlgete muutused.

2.2.2 *Portable Document Format* versioon

PDF-failivorming on loodud ettevõtte *Adobe Systems* poolt 1993. aastal ning on aja jooksul uute versioonidega mõnevõrra muutunud [3]. Üheks peamiseks sisemiseks muudatuseks on dokumendis sisalduvate piltide kokkupakkimise tehnoloogia, mis võimaldab dokumendi failimahtu vähendada. Suurimatest funktsionaalsuse muudatustest võib välja tuua versiooni 1.4, mis võimaldab dokumendi osade märgendamist [4]. See tähendab, et ka vaegnägijad saavad ekraanilugeja abiga aru, kas parasjagu loetav on tavaline tekst, peatüki pealkiri, tabel või mõnda muud tüüpi sisu.

Versioon 1.4 on ka kõige laialdasemalt kasutatav ehk kõige rohkem seadmeid suudab seda versiooni PDF-i korrektselt kuvada. Kõrgemat PDF-i versiooni kuvavad rakendused suudavad ka madalamat PDF-i versiooni kuvada. Võttes arvesse

vaegnägijate nõudmisi on mõistlik PDF-dokumendid väljastada vähemalt versiooniga 1.4. Hetkel ei ole rakenduses vajadust kõrgema PDF-versiooni järgi.

2.2.3 Piletimallide suurused

Kasutusel süsteem väljastab pileteid suuruses A4. See formaat on kõige levinum suurus printimiseks, kuna seda suurust printivad printerid on kõige laialdasemalt kättesaadavad. Kui inimene soovib piletit välja printida, siis kulub ära üks A4 paberileht. Siiski peab jääma süsteemil tulevikus suutlikus ka väiksemas suuruses pileteid väljastada. Näiteks võib nutitelefonis A4 formaadi kuvamine jääda veidi ebamugavaks ehk kasutaja peab piletikontrolliks nägema vaeva väikesel ekraanil sisu suurendamiseks.

3 Analüüs

Selles peatükis tuleb juttu võimalustest ja valikutest, kuidas lahendada eeltoodud probleeme.

3.1 Rakenduse struktuuriline jaotus

Siiani on PDF-pileti väljastamise funktsionaalsus jaotunud 3 erineva projekti vahel. Esiteks *Angularis* arendatud veeb, mille kaudu kasutaja küsib nupule vajutusega PDF-piletit. Seejärel Java baasil keskteenus, milles on kogu äriloogika ja suhtlus andmebaasiga. Viimaseks on Java baasil PDF-genereerimise mikroteenus, mille asendamist nüüd ka analüüsime.

Struktuurilises vaates täidab PDF-mikroteenus kahte erinevat ülesannet. Esiteks paneb vastavalt päringu andmetele kokku HTML-lehed. Teise ülesandena paneb teenus saadud HTML-lehtedest kokku PDF-dokumendi. Nähes et rakendus täidab erinevaid ülesandeid, siis tekib küsimus, kas rakenduse peaks jaotama eraldi osadeks. Lähtudes Robert C. Martini raamatu „Clean Architecture: A Craftman's Guide to Software Structure and Design" peatükist 27 „Services: Great and Small" võimaldab mikroteenuste struktuuriline lähenemine:

- kasutada erinevate teenuste arendamiseks erinevaid tehnoloogiaid [5]
- teenuseid käivitada ja ülal hoida erinevatel tehnoloogiatel ning serveritel, mis muudab serveri ressursside haldamise lihtsamaks [5]
- teenuseid käsitleda erinevate meeskondade vastutusalas [5]

Kolme väljatoodud eelise hulgast vähemalt kahest on PDF-mikroteenuse lahkulöömisel kasu. Erinevate meeskondade vastutusalasse siiski teenust anda ei saaks, kuna teenuse osad mõnel määral sõltuvad üksteisest. Näiteks teenuste vahelisel suhtlusel peab üks teenus teisega arvestama. Samuti kui üks teenus juhuslikult lakkab töötamast, siis

edukalt PDF-piletit enam genereerida ei ole võimalik. Võib järeldada, et teenuse eraldamisega kaheks osaks on pigem süsteemile kasu kui kahju. Seega tasub analüüsida HTML-lehe kokkupanekut ning sellest PDF-vormingus dokumendi genereerimist (edaspidi PDF-generaator) eraldi.

Lisaks on võimalus teenuseid eraldi hoides kasutada neid tulevikus mitmeks erinevaks otstarbeks. Näiteks lisaks piletitele võib PDF-generaatori teenust saada kasutada ka HTML-raportite PDF-vormingusse genereerimisel. Ettevõttes on käinud ka läbi mõte HTML-lehe teenust kasutada veebis HTML-pileti kuvamiseks. See tähendaks, et teenus tuleks kaitsta autentimisvahendiga, näiteks *Json Web Token*iga (JWT). Seda selleks, et inimesed ei saaks serverist kellegi teise pileti andmeid küsida.

3.2 PDF-generaatori analüüs

PDF-generaatori ülesandeks on HTML-lehest genereerida PDF-dokument. Veebiservereid, mis antud ülesannet täidavad, leiab *Google* otsingumootorist mitmeid. Paljud teenused küsivad kasutamise eest kuutasu ning tasuta pakutavad teenused ei ole tavaliselt kuigi töökindlad. Samuti ei ole kindel, et sellise teenuse pakkujad sisseantavat HTML-lehte oma huvides ära ei kasuta. Eesmärk oli süsteem ettevõttesiseselt ülesehitada. Äsja oli ettevõttes valminud ka uus *Kubernetes* haldussüsteem, kuhu *Dockeri* konteinerite abil on erinevates tehnoloogiates arendatud süsteeme lihtne seadistada.

Esialgu sai uuritud võimalusi seoses kasutusel oleva *Flyingsaucer* [6] Java teegiga. Seda lähemalt uurides selgus, et seesmiselt kasutab teek *iText 2* teeki. iText on vabavaraline teek, millega on võimalik PDF-dokumentide sisu töödelda. Teek on loodud Belgia tarkvaraarendaja Bruno Lowagie poolt 14. veebruaril 2000. aastal. Esialgu olid teegi versioonid avaldatud LGPL litsentsiga, mis võimaldas teeki vabamalt kasutada. Versioonist 5 avaldati iText aga juba AGPL-litsentsiga, mille tagajärjel oli teegi ärilisel eesmärgil (suletud lähtekoodiga) kasutamiseks vaja soetada luba [7]. Seega ei jäänud autor selle teegi kasutamise võimaluse peale pikemalt mõtlema ning tuli hakata otsima alternatiive, mis täidaksid soovitud ülesannet.

Võimalikke lahendusi otsides jäi mitmeid kordi nii *Stackoverflowst* kui ka *Redditist* silma soovitus kasutada vahendit nimega *wkhtmltopdf* [8]. *Wkhtmltopdf* on vabavaraline käsurea tööriist, millega on võimalik HTML-st genereerida PDF-fail. Tööriistale on loodud teeke väga paljudes erinevates keeltes.

Autor otsustas katsetada tööriista Golangi programmeerimiskeelele arendatud teeki [9], kuna sellel oli Githubis võrreldes teistele keeltele mõeldud teekidega rohkem tärne. Githubi tärnid näitavad koodivaramu populaarsust kasutajate hulgas [10]. Golang on tõusva populaarsusega programmeerimiskeel, mis on oma süntaksilt võrdlemisi lihtne. Populaarsete IDE-de arendaja *Jetbrains* on alates 2017. aastast läbi viinud küsitlust „The State of Developer Ecosystem", milles on Golang näidanud jõudsat kasvutrendi arendajate hulgas [11]-[14].

Koodinäide, millega autor *wkhtmltopdf* Golangi teeki katsetas, asub lisas 6. Katsetamiseks sai arendatud lihtne veebiserver, kuhu on võimalik saata päring, milles aadressi parameetrina on etteantud soovitud veebilehe aadress. Programm avab lehe ning genereerib HTML-st PDF-i, mis tagastatakse päringu vastusena. Testimise käigus sai selgeks, et vahend ei olnud lahendamiseks ideaalne. Näiteks sai proovitud genereerida PDF-i ühest *CSS grid* näidisest. *CSS grid* on kahedimensionaalne ruudustiku süsteem võimaldamaks määrata kasutajaliidese paigutust [15]. *Wkhtmltopdf* ei mõistnud *CSS gridi* funktsionaalsust ning kuvas paigutuse valesti. Näide on toodud lisas 7. Lähemalt uurides selgus, et nagu vahendi nimigi ütleb (wk ehk webkit) on veebilehe laadimiseks kasutusel *WebKit* mootor. *WebKit* on peamiselt tuntud Apple Safari veebilehitseja mootorina. Apple WebKit port ei ole aga vabalt kasutatav ning *wkhtmltopdf* kasutab *QTWebKit* porti. Põhjus, miks *wkhtmltopdf* ei ole võimeline *CSS grid* funktsionaalsust mõistma, peitubki *QTWebKit* mootoris. Nimelt ei ole seda alates 2015. aastast edasi arendatud [16] ning *CSS grid* on hiljem lisandunud CSS-i funktsionaalsus.

Alternatiivi *wkhtmltopdf* vahendile kaugelt otsida ei tulnud. Tööriista kodulehel on eraldi sektsioonis toodud välja soovitused PDF-dokumentide genereerimiseks [17]. Selgub et on arendatud *Node.js* teek *Puppeteer*, mis võimaldab kasutada *headless* Chrome veebilehitsejat PDF-dokumentide genereerimiseks. *Node.js* on serveris töötav Javascripti mootor. *Headless Chrome* on Google Chrome veebilehitseja režiim, millel ei

ole toimimiseks vaja graafilist kasutajaliidest. *Puppeteer* teek ongi loodud juhtima operatsioone *headless Chrome* veebilehitsejas. Leitud lahendus tundus olevat sobilik viis PDF-ide genereerimiseks, kuna Google Chrome on üks enim kasutatavaid veebilehitsejaid [18] ning laialdaselt kasutust leidvad CSS-i funktsionaalsused on toetatud. Lisaks saab eesrakenduse arendaja olla kindel, et kui tema kood toimib Google Chrome veebilehtitsejas, siis see toimib ka PDF-rakenduses.

Google Chrome kasutab PDF-i genereerimiseks Skia tagarakendust. Skia on vabavaraline 2-dimensionaalse graafika teek. Projekti haldab Google ning on kättesaadav BSD vaba tarkvara litsentsi alt [19].

3.3 Serveripoolse HTML-lehe koostamise teenuse analüüs

HTML-pileti kokkupanemiseks kasutatav Java-põhine *Apache Velocity* mallimootor ei sobi ettevõtte struktuuriga. Tarkvaraarendusega tegelev tööjõud on ettevõttes jaotatud eesrakenduse ja keskteenuse arendajateks. Arendajaid, kes arendaksid nii keskteenust kui eesrakendust, ettevõttes palju ei ole. Seega on ka Java ehk peamiselt keskteenuse arendamise tehnoloogia eesrakenduse arendajatele võõras.

Ettevõtte eesrakenduse arendajatele igapäevaselt kasutatav tehnoloogia on Angular. Angular on Javascripti raamistik, mis on mõeldud *Single Page Application (SPA*, eestik. üheleherakendus*)* kasutajaliideste ehitamiseks. Üheleherakendus ei tähenda, et rakendusel oleks vaid üks veebileht ning navigeerimine erineva sisu vahel oleks võimatu. Üheleherakendus tähendab hoopis, et rakendus laetakse esimese päringuga veebilehitseja mällu. Navigeerimisel ei laeta serverist enam uut lehte, vaid rakendus töötleb veebilehitseja mälus loogikat, et lehe sisu muuta ning vajadusel teeb päringuid serveri pihta, mis on tavaliselt JSON-vormingus. Selline lähenemine muudab esialgu veebirakenduse laadimise küll aeglasemaks, aga edaspidise sisu vahel navigeerimise kiiremaks. HTML-piletisisu väljastamiseks ei ole sisu pidev muutmine tähtis. Sisu seatakse malli kokkupannes ühe korra kui serverist andmed saabuvad. Seetõttu oleks vaja, et Angular töötaks serveri poolel ning ei laeks ennast iga uue pileti päringu korral kliendipoolse veebilehitseja (rakenduse puhul PDF-generaatori) mällu. HTML-i väljastamiseks serveri poolelt on loodud Angular Universal.

Angular Universal on projekt, mis laiendab Angulari raamistikku ning võimaldab genereerida HTML-lehti veebilehitseja asemel serveris. See lähenemine muudab pileti sisu esmakordse laadimise kiiremaks.

3.3.1 Alternatiivsed lahendused Angular Universalile

StackOverflow ja *Jetbrainsi* poolt läbi viidud arendajatele suunatud küsitlustest selgub, et Angular ei olegi kõige populaarsem Javascripti raamistik. Lisaks Angularile on rohkelt kasutusel ka React ning Vue raamistikud [21]-[22]. Kuigi Ridangos kasutatakse eesrakenduste arendamiseks Angulari, siis tasub kaaluda ka teisi üheleherakenduste arendamiseks kasutatavaid tehnoloogiaid.

React ja Vue raamistikel on samuti võimekus genereerida veebilehe sisu serveri poolel. Sarnaselt Angular Universalile on ka nende puhul võimalik üpriski väikese vaevaga sättida üles eraldi skriptid, mis võimaldavad rakendust jooksutada nii serveri kui ka kliendi poolel.

Kuna nii Reacti, Vue kui ka Angulari serveripoolne rakendus kasutab sisemiselt Express raamistikku, siis ei ole mõtet võrrelda nende jõudlust. Jõudlus sõltub raamistike enda keerukusest. Seega on serveri jõudlus sama ning vahe on ainult raamistike enda kiiruses. Express on minimalistlik Javascripti raamistik, mida kasutatakse põhjana väga paljude teiste Javascripti raamistike poolt, kuna Express raamistikul on paindlikud tööriistad, millega serveripoolset Javascripti rakendust ehitada.

Angular Universal on siiski kõigist kolmest kõige „universaalsem” lahendus. See tähendab, et sama koodibaasi ja lihtsa vaevaga saab rakendust jooksutada nii serveri kui ka kliendi poolel. See on oluline, sest sellisel juhul saavad eesrakenduse arendajad Angulari jooksutada ja testida harjumuspäraselt ning kliendipoolse rakendusega on muudatuse korral rakenduse ehitusaeg kiirem kui serveripoolsel.

4 PDF-generaatori teostus

Selles peatükis tuuakse detailselt välja PDF-generaatori mikroteenuse arendusprotsess.

4.1 Rakenduse prototüüp

Enne põhjalikumat rakenduse teostust tuli katsetada, kas analüüsis valitud tehnoloogia on ülesande lahendamiseks piisav. Prototüübi loomist alustas autor *pptr.dev* veebilehel väljatoodud näitest.

Esiteks tuli kasutada NodeJS paketihaldurit (edaspidi npm ehk Node Package Manager), et laadida alla Puppeteer teek, millega juhtida *headless Chrome* veebilehitsejat. Selleks kasutas autor käsureal järgmist käsku: *npm i puppeteer*. Selle tagajärjel tekkis prototüübi kausta *package.json* ja *package-lock.json* failid ning *node_modules* kaust. *Package.json* fail on projekti manifest, milles avaldatakse projekti versioon, nimi, sõltuvused, käivitusskriptid ja paljud muud kitsendused. Antud juhul määras *npm install* käsk JSON *dependency* massiivi järgneva sõltuvuse: *"puppeteer": "^8.0.0"*. See tähendab, et nõutud on Puppeteer teegi versioon 8.0.0 ning ^ märk tähendab, et jooksutades käsku *npm update* ei uuendataks paketi *major* (kõige vasakpoolsem number) versiooni, vaid ainult *minor* (keskmine number) ja *patch* (kõige parempoolsem number) versiooni. *Package-lock.json* täpsustab detailselt ära, millist paketi versiooni hetkel kasutada. Selle mõte on, et pakette *npm update* käsuga uuendades on võimalik naasta viimase toiminud paketi versiooni juurde. *Node_modules* kaust hoiab endas allalaetud sõltuvaid teeke. Lisaks Puppeteer teegile on sinna laetud ka kõik Puppeteer teegi sõltuvused. Puppeteer teegile ja sõltuvustele lisaks laeti alla Chromium veebilehitseja versioon, mis ühildub Puppeteer teegi versiooniga. Chromium on avatud lähtekoodiga projekt, millel baseerub Google Chrome veebilehitseja.

Olles sõltuvused allalaadinud oli võimalik edasi liikuda prototüübiga. Tuli luua kood, millega teegi toimivust katsetada. Selleks tegi autor uue faili, mille sisu oli järgmine:

```
1  const puppeteer = require('puppeteer');
2
3  (async () => {
4    const browser = await puppeteer.launch();
5    const page = await browser.newPage();
6    await page.goto('https://gridbyexample.com/examples/code/example5.html', {
7        waitUntil: "networkidle0"
8    });
9    await page.pdf({ path: 'example.pdf', format: 'a4', printBackground: true });
10
11  await browser.close();
12 })();
```

Joonis 1: PDF-generaatori prototüüp

Prototüübi koodist on näha, et esimesel real luuakse Puppeteer objekt. Rida 3 on ümbritsetud asünkroonse funktsiooniga, et funktsiooni sees olevad käsud, mille ees on *await* võtmesõna, käivituksid järjestikuses korras. Kui asünkroonset funktsionaalsust ei kasutaks, siis võib juhtuda näiteks see, et PDF genereeritakse ennem valmis kui veebileht jõuab ära laadida. Tulemuseks on tühi PDF-pilet. Rida 4 avab veebilehitseja ning rida 5 avab veebilehitsejas uue vahelehe. Rida 6 *goto* funktsioon juhib veebilehitseja avama etteantud veebilehte ning täpsustab, mis tingimustel saab lehe laadimise lugeda lõpetatuks. *Networkidle0* tähendab, et veebilehte laadides peab Puppeteer tegema kindlaks, et viimase 500 millisekundi jooksul ei ole pooleli ühtegi võrguühendust [23]. Kui kood jõuab reani 9 on veebileht laetud ning võib asuda lehte PDF-vormingusse genereerima. Funktsioonile *pdf* saab kaasa anda erinevaid genereerimise argumente. Antud juhul on kaasa antud *path*, mis määrab ära väljastatava PDF-vormingus faili nime ja asukoha failisüsteemis. Veel on kaasa antud *format*, mis täpsustab, et PDF peab olema A4 formaadis, ja *printBackground*, mis ütleb, et lehe ning lehe elementide taust tuleb samuti PDF-i kaasa anda.

Prototüüpi saab käivitada käsuga: *node example.js.* Selle tagajärjel tekib programmi kausta *example.pdf* fail. Prototüüp töötab etteantud CSS *grid* näitega perfektselt. Seega on valitud tehnoloogia piisav rakenduse edasiseks teostamiseks.

4.2 Rakenduse kirjeldus

PDF-generaatori teostamiseks oli vajalik veebiserver. Esialgu tundus mõistlik kasutada minimalistlikku Express raamistikku, kuna valmiv rakendus ei olnud eriti suuremõõtmeline. Kogenenud eesrakenduse arendajaga veidi arutledes sai aga

otsustatud NestJS raamistiku kasuks. Võib öelda, et NestJS raamistik loob Express raamistiku peale abstraktsioonikihi, millega on võimalik kiiremini rakendust arendada. Kogu Express raamistiku funktsionaalsust saab endiselt kasutada. Samuti on NestJS raamistikus vaikimisi kasutusel Typescript, mis võimaldab kirjutada tüübitud Javascripti ning vähendada vigu arendamisel. Kõige tipuks on NestJS algusest peale arhitektuuriliselt struktureeritud, mis jätab võimalused lihtsamini kirjutada ühikteste. NestJS raamistiku struktuur on inspireeritud Angulari struktuurist, mis on kindlasti tervitatav asjaolu Angulari arendajatele.

NestJS dokumentatsiooni põhjal paigaldas autor *nest* käsurea liidese: *npm i -g @nestjs/cli*. Raamistiku käsurea liidesega on võimalik lihtsasti luua projekti struktuur. Uue projekti jaoks tuli kasutada käsku: *nest new backend-pdf-generator.* Selle peale loodi kogu projekti struktuur, mis oli järgmine:

```
.
├── nest-cli.json
├── node_modules
├── package.json
├── package-lock.json
├── README.md
├── src
│   ├── app.controller.spec.ts
│   ├── app.controller.ts
│   ├── app.module.ts
│   ├── app.service.ts
│   └── main.ts
├── test
│   ├── app.e2e-spec.ts
│   └── jest-e2e.json
├── tsconfig.build.json
└── tsconfig.json
```

Joonis 2: Nest käsurea liidese poolt loodud projekti struktuur

Package.json failide ja *node_modules* kausta ülesannetest oli juba eelmises peatükis juttu, aga ülejäänud on veel selgitamata. *Nest-cli.json* on konfiguratsiooni fail *nest* käsurea liidesele. *Tsconfig* failid on loodud Typescripti seadistamiseks. *Test* kaustas asuvad rakenduse testid. Ja viimaks *src* kaust, kus asub rakenduse kood, mis on jaotatud mooduliteks, kontrolleriteks ja teenusteks. Rakenduse alguspunktiks on *main.ts* fail, milles kutsutakse välja server ning määratakse serveri parameetrid. Näiteks mis pordil server jookseb. Kontrolleri fail on mõeldud päringute vastuvõtmiseks ja neile vastamiseks. *Service* ehk teenuse fail on mõeldud äriloogiliseks tegevuseks, mida võib saada kutsuda välja erinevatest kontrolleritest.

Edasi tulebki luua kontroller, mis suudaks vastu võtta põhiteenusest tulevat päringut. Uue kontrolleri ning teenuse struktuuride loomiseks kasutas autor käske: *nest g controller pdf* ja *nest g service pdf.* Selle peale lisati rakenduse *src* kausta *pdf* kaust, mille struktuur oli järgmine:

```
.
├── pdf.controller.spec.ts
├── pdf.controller.ts
├── pdf.service.spec.ts
└── pdf.service.ts
```

Joonis 3: *Nest* liidesega loodud pdf kausta struktuur

Nii kontrollerile kui ka teenusele loodi *spec* fail, mis on mõeldud vastava kontrolleri või teenuse ühiktestimiseks kasutades vaikimisi *Jest* testimise raamistikku.

Autor täiendas PDF-kontrollerit arenduse käigus järgnevalt:

```
1  @Controller('pdf')
2  export class PdfController {
3      constructor(private readonly pdfService: PdfService) {}
4
5      @Get('generate')
6      async generatePdfFromUrl(@Req() request: Request, @Res() response: Response): Promise<any>  {
7          const logPrefix = this.generatePdfFromUrl.name;
8          const url = request.headers['x-url'];
9          if (!url || Array.isArray(url)) {
10             Logger.log('Bad request: x-url header is missing', logPrefix);
11             return response.sendStatus(HttpStatus.BAD_REQUEST);
12         }
13         Logger.log('Incoming url: ' + url, logPrefix);
14         let buffer: Buffer;
15         try {
16             buffer = await this.pdfService.generatePdfFromUrl(url);
17         } catch (e) {
18             Logger.log('PDF generation for page (' + url + ') failed with error: ' + e, logPrefix);
19             return response.sendStatus(HttpStatus.UNPROCESSABLE_ENTITY);
20         }
21
22         response.set({
23             'Content-Type': 'application/pdf',
24             'Content-Length': buffer.length,
25         });
26         return response.end(buffer);
27     }
28 }
```

Joonis 4: PDF-generaatori kontroller

Kontrolleri kood määrab annotatsioonidega aadressi, kust saab GET-päringuga PDF-dokumenti genereerida. Klassi konstruktor teeb võimalikuks *PdfService*-i ehk teenuse loogika kasutamise. Rida 9 – 12 valideerib sissetulevat päringut. Päringu päises peab alati kaasa saatma veebilehe aadressi, millest PDF-i genereerida. Rida 16 tegeleb PDF-i

genereerimisega. See on asetatud erindiga tegelevasse koodiplokki, kuna PDF-i genereerimisel võib tekkida ootamatusi ning need tuleb päringu vastuses ka korrektselt kajastada. Viimane osa kontrolleri koodist (rida 22 – 26) tegeleb päringu vastuse ja vastuse sisu tüübi seadmisega.

PdfService, ehk klass mis avab etteantud veebilehe ja muudab selle PDF-vormingus dokumendiks, on täiendatud järgmiselt:

```
1  @Injectable()
2  export class PdfService {
3      async generatePdfFromUrl(url: string): Promise<Buffer> {
4          const browser = await puppeteer.launch({
5              headless: true,
6              // need to launch no-sandbox, since alpine container won't start chrome otherwise
7              args: ['--no-sandbox'],
8          });
9          const page = await browser.newPage();
10         const response = await page.goto(url, {
11             // need to wait before page is fully loaded
12             waitUntil: 'networkidle0',
13         });
14         if (response == null || response.status() != HttpStatus.OK) {
15             throw new Error('Failed to navigate to page. Http status code: ' + response.status());
16         }
17         const buffer = await page.pdf({ format: 'A4', printBackground: true });
18         await browser.close();
19         return buffer;
20     }
21 }
```

Joonis 5: PDF teenuse kood

Teenuse koodi kirjutamiseks oli vaja allalaadida Puppeteer teek, mida on juba demonstreeritud eelmises peatükis. Real 1 on NestJS annotatsioon *Injectable*, et käesolevat klassi oleks võimalik sisestada teistesse klassidesse konstruktori abiga (nii nagu seda on kontrolleris ka tehtud). *GeneratePdfFromUrl* funktsioon võtab sisse veebilehe aadressi sõne kujul ning tagastab PDF-faili puhvri kui ei teki viga. Rida 4 avab veebilehitseja. Avamisel antakse kaasa parameetrid *headless* ja *args*. *Headless* parameeter tuleb kaasa anda, et veebilehitsejat ei avataks graafilises režiimis. *Args* võimaldab kaasa anda veebilehitseja avamise argumente. Lähemalt miks oli vaja kaasa anda argument *no-sandbox* tuleb juttu rakenduse konteinerisse paigaldamise peatükis. Rida 9 ja 10 avavad uue vahelehe ja laevad etteantud veebilehe nii nagu juba prototüübis kirjeldatud. Rida 14 kontrollib, kas veebilehe laadimine õnnestus. Kui ei õnnestunud, siis visatakse erind ja olemasolul kirjelduses ka päringu vastuse kood. Kui

lehe laadimine oli edukas, siis genereeritakse A4 formaadis PDF-dokumendi puhver. Viimaseks suletakse avatud veebilehitseja ning tagastatakse saadud puhver.

4.3 Rakenduse testimine

Selles peatükis tuleb lähemalt juttu, kuidas toimus rakenduse testimine.

4.3.1 Ühiktestimine

Ühiktestimiseks kasutas autor NestJS-ga juba kaasas olevat Jest testimise raamistikku. *Pdf.controller.ts* fail sai testitud 100-protsendilise katvusega. Testitud sai päise kaasa andmise funktsionaalsust, mille puudumisel oli päringu vastuses HTTP-staatuskood 400 *Bad Request*. Edasi sai testitud *PdfService* teenuse välja kutsumist ning selle õnnestumise korral vastuse staatuskoodi (200), sisu ja päiseid. Viimaks kattis autor testiga ka juhu kui *PdfService* viskas erindi. Sel juhul tagastati vastav HTTP-staatuskood.

```
it('should call pdf service', async () => {
      requestMock.headers['x-url'] = 'website';
      let buffer = Buffer.from('test');
      const spy = jest.spyOn(pdfServiceMock, 'generatePdfFromUrl').mockResolvedValue(buffer);

      await controller.generatePdfFromUrl(requestMock, responseMock);

      expect(spy).toHaveBeenCalledTimes(1);
      expect(responseStatus).toBe(HttpStatus.OK);
      expect(responseHeaders).toHaveProperty('Content-Type', 'application/pdf');
      expect(responseHeaders).toHaveProperty('Content-Length', buffer.length);
      expect(responseBody).toBeDefined();
   });
```

Joonis 6: PDF-generaatori ühiktesti näide

PdfService koodi nii lihtne testida ei olnud, kuna kasutusel oli väline *Puppeteer* teek. Testide jooksutamiseks tuli kasutada käsku: *jest*. Koodi kaetavust nägi käsuga: *jest –coverage*. See käsk käivitas testid ning analüüsis mis osad ja kui suur protsent koodist olid testitud.

4.3.2 Koormuse testimine

Koormuse testimiseks on autor kasutanud vabavaralist vahendit nimega K6. Seda sai kasutatud ka PDF-generaatori rakenduse koormuse katsetamiseks. K6 jaoks kirjutatud

test asub lisas 8. Jõudluse testi suunas autor vastu Google avalehte. Jõudlustesti pikkuseks määras autor 1 minuti ning virtuaalseid kasutajaid, kes korraga päringuid sooritasid 10.

```
running (1m02.3s), 00/10 VUs, 213 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  1m0s

  ✓ successful response

  checks.....................: 100.00% ✓ 213  ✗ 0
  data_received..............: 22 MB   346 kB/s
  data_sent..................: 26 kB   423 B/s
  http_req_connecting........: avg=4.71µs   min=0s     med=0s     max=179.78µs p(90)=0s     p(95)=0s
  http_req_duration..........: avg=2.87s    min=2.35s  med=2.87s  max=3.86s    p(90)=3.04s  p(95)=3.24s
  http_req_tls_handshaking...: avg=0s       min=0s     med=0s     max=0s       p(90)=0s     p(95)=0s
  http_req_waiting...........: avg=2.87s    min=2.35s  med=2.87s  max=3.86s    p(90)=3.04s  p(95)=3.24s
  http_reqs..................: 213     3.419303/s
  iteration_duration.........: avg=2.87s    min=2.35s  med=2.87s  max=3.86s    p(90)=3.04s  p(95)=3.24s
  iterations.................: 213     3.419303/s
  vus........................: 2       min=2  max=10
  vus_max....................: 10      min=10 max=10
```

Joonis 7: PDF-generaatori K6 koormustesti tulemused

Testi tulemustega võib rahul olla. Eesmärk oli rakendust koormata ja vaadata ega ei teki anomaaliaid, kus PDF-i genereerimine võiks koormuse all katki minna. Kõik päringud said korrektselt Google avalehest PDF-dokumendi kätte.

4.4 Rakenduse paigaldamine *Docker*i konteinerisse

Ridango teenused on 2021. aasta aprilliks üle läinud virtuaalmasinatelt *Docker*i konteineritele. Rakendus tuli paigaldada konteinerisse. Esialgu tutvus autor *Puppeteer* teegi dokumentatsiooniga ning selgus, et konteinerisse paigaldamine ei olegi nii lihtne kui mõne tavalise NodeJS rakenduse puhul [24]. *Puppeteer* teegiga laetakse alla Chromiumi veebilehitseja, mille versioon on sõltuv teegi versioonist. *Docker*i baaskonteineriks on tavaliselt saanud valitud Alpine Linux, mis on populaarne oma minimalistliku suuruse poolest. Teegiga kaasas käiv Chromium aga selles Linuxi distributsioonis ei käivitunud, kuna puudusid ühilduvad süsteemi teegid. Peale pikka aega lahenduse kallal katsetamist proovis autor Alpine Linuxi paketihaldurist kättesaadavat Chromiumi versiooni. Puppeteer teek tuli viia allapoole versioonile 5.3 [25], et ta ühilduks Alpine Linux 3.12 paketihaldurist saadava Chromium versiooniga 86 [26].

Lisaks tuli koodist veebilehitseja avamisel kaasa anda argument *no-sandbox* [27]. Nimelt loob Chrome veebilehitseja turvalisuse tagamiseks operatsioonisüsteemi protsesse, mis käivituvad piiratud kasutusõigustega [28]. Nende protsesside käivitamiseks aga Alpine Linux baaskonteiner seadistatud ei olnud. Autor läks hetkel kergema vastupanu teed ning käivitas veebilehitsejat ilma *sandbox* protsessita, kuna oli teada, et ettevõtte välist sisu ei olnud tarvis generaatoril laadida.

Lisaks tuli arenduse paigaldamise automatiseerimiseks seada üles ettevõttes kasutatava koodivaramu *Bitbucket pipeline*-d (skriptid, mis automatiseerivad rakenduse paigaldamist).

5 Angular piletimalli rakenduse teostus

Selles peatükis tuuakse detailselt välja piletimallide väljastamise mikroteenuse arendusprotsess.

5.1 Angulari rakenduse kirjeldus

Autor alustas tavalise (kliendipoolse) Angulari rakenduse arendamist. Esialgu ei olnud tarvis serveripoolset moodulit, kuna seda oli võimalik paigaldada ja kohandada hiljem.

Angulari rakenduse loomiseks tuli paigaldada Angulari käsurea liides. Käsurea liidese abil oli võimalik luua uus projekt: *ng n backend-angular-template-renderer*. Selle tagajärjel tekkis Angulari rakendusele omane struktuur, kus Angulari rakendus paiknes *src* kaustas. Edasi genereeris autor uue komponendi Elroni piletimallile: *ng g c elron/invoice.* Selle tagajärjel tekkis *src/app* kausta uus kaust *elron*, mille sees oli piletimalli tüübi järgi kaust *invoice*. Selles kaustas asub Elroni piletimalli komponent. Veel tuli luua uus *service* ehk teenus, mis võimaldaks sissetuleva tokeni järgi pärida keskteenusest piletimallile andmeid: *ng g s services/PdfData.* Selle tagajärjel oli loodud järgnev Angulari struktuur:

```
.
├── app.component.html
├── app.component.scss
├── app.component.spec.ts
├── app.component.ts
├── app.module.ts
├── app-routing.module.ts
├── elron
│   └── invoice
│       ├── invoice.component.html
│       ├── invoice.component.scss
│       ├── invoice.component.spec.ts
│       └── invoice.component.ts
└── services
    ├── pdf-data.service.spec.ts
    └── pdf-data.service.ts
```

Joonis 8: Angulari rakenduse *app* kausta struktuur peale põhikomponentide genereerimist

Pdf-data.service.ts tagastas esialgu andmete imitatsiooni ning põhiteenuse pihta päringuid ei sooritanud. Niimoodi oli lihtsam piletimalli arendada. Piletimall asub *invoice.component.html* failis ning see sai peamiselt rida realt ettevõtte eelmisest *Velocity* mallimootoriga teenusest Angulari mallimootorisse ümber kirjutatud.

5.2 Piletimalli andmete voog

Piletimalli andmeid tuli küsida keskteenusest saadetud JWT põhjal. Selline näeb välja *pdf-data.service.ts* teenus:

```
1  export class PdfDataService {
2      private baseUrl: string;
3      constructor(private http: HttpClient, appConfig: AppConfigService) {
4          this.baseUrl = appConfig.getConfig().API_URL;
5      }
6      fetchPdfData(token: string, requestIdentifier: number): Observable<Invoice> {
7          const url = `${this.baseUrl}/v2/1/pdf/data/request`;
8          //const url = '/assets/data-mock.json';
9          const headers = new HttpHeaders({ 'X-app-token': token });
10          let invoice = new Invoice();
11          return this.http
12              .get<InvoiceResponseDTO>(url, { headers: headers })
13              .pipe(
14                  map((request) => {
15                      invoice.tickets = request.json;
16                      invoice.lang = request.lang;
17                      invoice.tickets.forEach((ticket) => {
18                          if (ticket.currency && ticket.currency != 'EUR') {
19                              invoice.currency = ticket.currency;
20                          }
21                          invoice.total_price_sum += ticket.price!;
22                          invoice.vat_sum += ticket.vat!;
23                      });
24                      invoice.without_vat = invoice.total_price_sum - invoice.vat_sum;
25                      return invoice;
26                  })
27              );
28      }
29  }
```

Joonis 9: Pdf-data.service.ts kood

Keskteenuse aadressi sisselugemine käib läbi *AppConfig* teenuse. Aadress asub *settings.json* failis Angulari *assets* kaustas, kust on võimalik rakenduse jooksmise ajal see sisse küsida. Vajadus sätteid jooksvalt sisselugeda tekkis rakendust läbi *Ansible* konfiguratsiooni tööriista paigaldades. Nimelt ei ole sellisel juhul vajadust konfiguratsiooni muutmisel uuesti rakendust ehitada. Rida 9 määrab ära keskteenuse pihta tehtava päringu päise, milleks on sisse tulnud JWT. Selle puudumisel ei ole

keskteenusel õigust andmeid rakendusele tagastada. Rida 11 kutsub keskteenuse päringu välja ning määrab ära, mis kujule vastust töödelda. Samuti peab hetkel keskteenusest tulevaid andmeid veidi ümbertöötlema, et saaks piletite pealt kätte kõigi piletite summa ja käibemaksu. Need andmed peaks tulevikus juba keskteenuse poolt tagastatama. Hetkel seda ei tehta ning sujuvaks üleminekuks eelmiselt süsteemilt ei hakka autor seda ka praegu ümber tegema.

5.3 Serveripoolne Angulari moodul

Angulari serveris jooksutamiseks on vajalik veebiserver. Angulari ametlikus dokumentatsioonis soovitatakse kasutada Express mootoriga moodulit [29]. Ühtlustamaks rakenduste struktuuri tundus aga mõistlik ka Angulari puhul kasutada väikese abstraktsioonikihiga NestJS Angulari moodulit [30].

Mooduli paigaldamiseks kasutas autor projektis järgmist käsku: *ng add @nestjs/ng-universal*. Angulari serveri rakenduse ehitamiseks ja käivitamiseks tuli kasutada käsku: *npm run build:ssr && npm run serve:ssr*. Endiselt töötas ka kliendipoolne Angular käsuga: *ng s*. Probleem tekkis serveripoolse rakendusega konfiguratsiooni küsimisel *assets* kaustast. Serveripoolsele rakendusele ei piisa relatiivsest aadressist nagu näiteks *assets/settings.json*. Konfiguratsiooni laadimiseks on vajalik absoluutne aadress [31]. Selleks, et rakendus toimiks korrektselt nii kliendi veebilehitsejas kui ka serveris, oli vajalik luua päringu *interceptor* ehk vahelüli, mis muudaks kõigis rakenduse päringutes relatiivse aadressi absoluutseks aadressiks. Selleks kasutas autor Brian Bishopi blogi postituses välja toodud lahendust [32].

5.4 Tõlgete haldamine

Üheks probleemiks eelmise Ridango PDF-teenusega oli tõlgete haldus. Uue tõlke jaoks pidi varem looma eraldi mallifaili. Angulariga on tõlgete haldamine lihtsam. Autor kasutas ettevõtte Angular veebide eeskujul rakenduses *ngx-translate/core* teeki. Selle teegiga on võimalik lihtsasti mallis defineerida märksõnad, mida on vaja tõlkida ning sõna järel kutsuda välja *translate pipe* ehk Angulari mallist välja kutsutav funktsioon, mis asendab etteantud sõne vastava tõlkega. Tõlkefailid on JSON-vormingus, kus

tõlkefaili nimi ütleb ära, mis keele tõlkega parasjagu tegemist on. Failide sisuks on mallis kasutatav märksõna võtmena ning tõlge võtme väärtusena.

```
<div class="col-item w11 bold">
   {{ 'NOTE_TO_PASSENGER' | translate }}
</div>
...
et.json sisu:
{
   "NOTE_TO_PASSENGER": "Reisija meelespea"
}
```

Joonis 10: Tõlke näide rakendusest

6 Rakenduste integratsioon ja tulemused

Selles peatükis on kirjeldatud vajaminevaid muudatusi arendatud süsteemi kasutuselevõtuks ning koostatud ka jõudluse võrdlus kasutusel oleva ja valminud rakenduse vahel.

6.1 Keskteenuse muudatused

Java-põhiselt PDF-teenuselt üleminekuks töö käigus arendatud süsteemile tuli teha keskteenuses mõningaid muudatusi. Võrreldes eelmise süsteemiga ei tulnud enam piletiandmeid saata PDF-teenusesse otse, vaid Angular käis neid keskteenusest väljastatud JWT põhjal ise küsimas. Selleks tuli teha keskteenusesse uus liides, mis tagastas JWT põhjal andmebaasist piletiandmeid. Samuti tuli muuta keskteenuses päringut, millega PDF-dokumenti küsitakse. Andmete saatmise asemel tuli genereerida JWT, mis sisaldas endas päritava PDF-dokumendi keelt ja ostukorvi numbrit.

6.2 Jõudluse võrdlus eelneva ja valminud teenuse vahel

Jõudluse testimine toimus vastu keskteenust, mis vastavalt päringule kutsus välja ettevõtte Java-põhise PDF-teenuse või töö käigus arendatud uue teenuse. Jõudluse võrdlemiseks tuli andmebaasist tulevaid piletiandmeid imiteerida, kuna lokaalset andmetega baasi on ettevõtte keskteenuse jaoks raske luua. Kaugühendusega andmebaasi puhul ei pruugi võrdlus olla õiglane andmebaasi ühenduse kiiruse muutlikuse tõttu.

Autor kasutas testide jooksutamiseks K6 testimise vahendit. Kirjutatud test pani 10 virtuaalset kasutajat 1 minuti jooksul keskteenuse pihta PDF-pileteid küsima. Teste jooksutati ükshaaval, mitteparalleelselt. Kasutusel oleva ja uue süsteemi pihta jooksutati teste vaheldumisi. Mõlemat PDF-pileti teenust testiti 3 korda ning arvutati keskmine

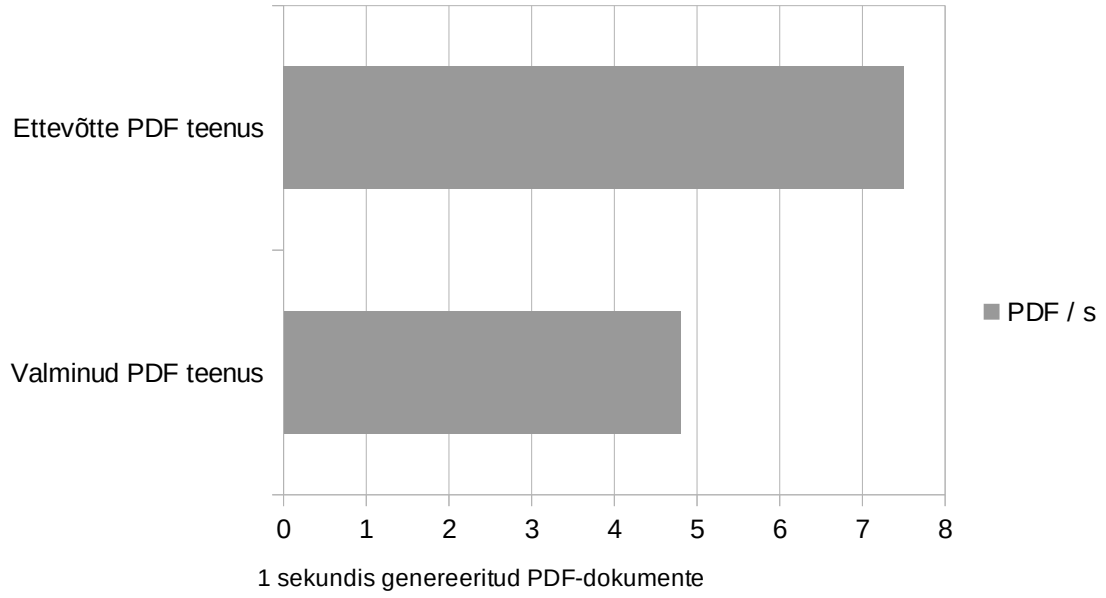
tulemus. Testimise tulemusel saadi mitu Elroni PDF-piletit jõuti süsteemiga 1 minuti jooksul 10 kasutaja poolt genereerida.

Testitulemuste väljundi näited on toodud lisas 9. Katsete tulemused on toodud järgnevas tabelis:

Tabel 1: Teenuste jõudlustesti katsete tulemused

	Katse 1	Katse 2	Katse 3
Eelmine lahendus	511	452	383
Uus lahendus	292	309	267

Keskmiselt jõuti Java baasil lahenduse korral genereerida 449 PDF-piletit ning uue lahenduse korral 289 PDF-piletit.



Joonis 11: Teenuste poolt 1 sekundis genereeritud PDF-dokumentide graafik

Tulemustest selgus, et lahendust võiks saada veel optimiseerida. Peale rakenduste osade eraldi mõõtmist selgus, et PDF-generaatori mikroteenusesse oli sattunud viga. Nimelt sai seatud Puppeteer teegis HTML-lehe laadimisel parameeter *waitUntil: networkIdle0*, mis ootas lehe laadimist pool sekundit peale võrguliikluse lõppemist. Seda parameetrit oli esialgu vaja kliendipoolse Angulari rakenduse korral, et Javascript jõuaks veebilehitsejas loogika ära laadida. Serveripoolse Angulari rakenduse korral võib seada

parameetri väärtuseks *load*, mis ootab ära lehe täieliku laadimise [33-34]. Muudatuse järel sai teenusele tehtud uued jõudluse katsed.

Tabel 2: Jõudluse tulemused peale PDF-generaatori parandust

	Katse 1	Katse 2	Katse 3
Eelmine lahendus	511	452	383
Uus lahendus, esialgne	292	309	267
Uus lahendus peale parandust	371	356	369

Uue teenuse keskmine tulemus oli nüüd keskmiselt 365 piletit minutis.

Tulemuste osas leiab autor, et lahendust võib saada veel optimiseerida. Siiski oli kavandatud teenuse puhul mõningane jõudluse kadu oodatav, sest ühe monoliitse PDF-teenuse asemel sai tehtud 2 mikroteenust. Tuleb arvestada, et mikroteenustevaheline suhtlus tõstab märgatavalt pärigule kuluvat aega. Teine jõudlust mõjutav faktor on kindlasti Angulari kasutamine HTML-pileti genereerimisel ning veebilehitsejas selle avamine PDF-generaatori poolt. Eelmises lahenduses genereeris Velocity mallimootor HTML-lehe valmis ning suunas edasi F*lyingsaucer* teegile PDF-vormingusse muutmiseks.

6.3 Keskteenuse ja eesrakenduse suhtlus ning PDF-pileti väljastamine

Arendatud süsteem ei muutnud suhtlust keskteenuse ja eesrakenduse vahel. PDF-piletite genereerimine on endiselt keskteenuse taga ning välisest maailmast ligipääs puudub. Keskteenus väljastab sisseloginud kasutajale PDF-pileti vaid juhul kui kasutaja on küsitud ostukorvi omanik. Anonüümsete ehk autentimata kasutajate ostudele väljastatakse peale ostu keskteenuse poolt JSON Web Token, mis pannakse PDF-piletit küsides eesrakendusest päringule kaasa. Keskteenus verifitseerib, et token oleks väljastatud tema poolt ning et tokenis sisalduks küsitud ostukorvi number. Selle järgi saab keskteenus aru, kas PDF-piletit võib väljastada või mitte. Samuti saadetakse PDF-pilet peale ostu manusena ostja e-posti aadressile.

7 Hinnang tehtud tööle ning edasiarenduse võimalused

Valminud töö näitab, et PDF-i piletimalle on võimalik genereerida ka ettevõtte eesrakenduse arendajale mugava tehnoloogiaga, milleks on Ridango AS puhul Angular raamistik. Siiski peab arvestama, et jõudluse poolest ei pruugi lahendus olla samal tasemel olemasoleva Java baasil rakendusega. Valminud lahenduse kasuks räägib asjaolu, et kasutusel olev süsteem ei ole enam kuigi kaasaegne ning piletimallide arendamisele kulub kauem aega kui valminud süsteemiga. Seda juba seetõttu, et Angulariga on võimalik hõlpsasti luua taaskasutatavaid komponente, mida Velocity mallimootoriga võimalik teha ei ole. Näiteks luua baaskomponent, millega kuvada piletimallil piletite loendit.

Edasiarenduse vaates võib proovida jõudlust parandada andmevoo muutmisega. Näiteks võiks kogu piletiinfo juba JWT sees kaasa anda. See lähenemine vähendaks keerukust ühe päringu võrra. Teise edasiarendusena võiks keskteenusesse arendada PDF-i päringule *cache-i* ehk vahemälu PDF-pileti vastuse salvestamiseks. Ettevõttel on andmete vahemällu salvestamiseks kasutusel Redise server. Veel üheks edasiarenduseks võiks korrastada keskteenusest saadetavad andmed. Andmeedastustobjekt (DTO) jäi samaks, mis Java projektiga ning sisaldab andmeid, mida PDF-dokumendil tegelikult ei kasutata.

Tulevikus võiks arendatud Angulari projekti viia üle ettevõtte veebirakendustega samasse monorepositorriumisse (ühtesesse koodivalamusse). Selline lähenemine võimaldaks kasutada PDF-piletimallide arendamisel ka veebis kasutatavaid Angulari komponente.

8 Kokkuvõte

Lõputöö raames valmis ettevõttele Ridango AS uus PDF-piletite genereerimise süsteem. Valminud süsteem on kaasaegsem ning paremini arendatav kui eelmine süsteem, kuna eesrakenduse arendajal on võimalik veebilehitsejas ka lõpptulemust näha. Eelmise lahendusega oli see raskendatud, kuna rakendus genereeris serveris HTML-st koheselt PDF-dokumendi. Valminud süsteemi saab Angulari raamistikuga arendada kui veebilehte.

Valminud lahendus täitis kõik töö alguses püstitatud eesmärgid. Piletimalle arendades on võimalik kasutada CSS-stiilikeele atribuute, mida mõistavad kaasaegsed veebilehitsejad. Eesrakenduse arendaja ei pea enam otsima CSS2.1 standardile vastavaid atribuute, mida süsteem lugeda oskaks. Piletimallide tõlgete haldus on koondunud JSON-failidesse ning enam ei ole vaja iga keele jaoks luua eraldi piletimalli. Angular võimaldab erinevate piletimallide jaoks luua baaskomponente, mis vähendab uue piletimalli arendamiseks kuluvat tööaega. Samuti ei pea CSS asetsema enam HTML-ga samas failis.

Kasutatud kirjandus

[1] „CSS modules status,” [Võrgumaterjal]. Saadaval: https://developer.mozilla.org/en-US/docs/Archive/CSS3#css_modules_status. Kasutatud: 17.02.2021.

[2] „Gradle vs Maven Comparison,” [Võrgumaterjal]. Saadaval: https://gradle.org/maven-vs-gradle/. Kasutatud: 30.01.2021.

[3] „The application/pdf Media Type,” [Võrgumaterjal]. Saadaval: https://tools.ietf.org/html/rfc8118#section-2. Kasutatud: 07.04.2021.

[4] „PDF versions,” [Võrgumaterjal]. Saadaval: https://www.prepressure.com/pdf/basics/version. Kasutatud: 07.04.2021.

[5] Robert C. Martin, Clean Architecture: A Craftman’s Guide to Software Structure and Design. 2017. [Võrgumaterjal]. Saadaval: http://prof.mau.ac.ir/images/Uploaded_files/Clean%20Architecture_%20A%20Craftsman%E2%80%99s%20Guide%20to%20Software%20Structure%20and%20Design-Pearson%20Education%20(2018)[7615523].PDF. Kasutatud: 02.03.2021. Peatükk 27.

[6] „flyingsaucer,” [Võrgumaterjal]. Saadaval: https://github.com/flyingsaucerproject/flyingsaucer. Kasutatud: 21.02.2021.

[7] B. Lowagie, „Bio Bruno Lowagie,” [Võrgumaterjal]. Saadaval: https://www.lowagie.com/bio. Kasutatud: 21.02.2021

[8] „[Question] How to convert HTML to PDF file,” [Võrgumaterjal]. Saadaval: https://www.reddit.com/r/webdev/comments/5ihsnh/question_how_to_convert_html_to_pdf_file/db8abg6?utm_source=share&utm_medium=web2x&context=3. Kasutatud: 21.02.2021.

[9] S. Klippert, „go-wkhtmltopdf,” [Võrgumaterjal]. Saadaval: https://github.com/SebastiaanKlippert/go-wkhtmltopdf. Kasutatud: 24.02.2021.

[10] „Github stars – useful metric for what,” [Võrgumaterjal]. Saadaval: https://opensource.stackexchange.com/questions/5110/github-stars-is-a-very-useful-metric-but-for-what#:~:text=Users%20on%20the%20GitHub%20website,on%20with%20the%20repo%20later. Kasutatud: 24.02.2021.

[11] „The State of Developer Ecosystem in 2017,” [Võrgumaterjal]. Saadaval: https://www.jetbrains.com/research/devecosystem-2017/. Kasutatud: 21.02.2021.

[12] „The State of Developer Ecosystem in 2018,” [Võrgumaterjal]. Saadaval: https://www.jetbrains.com/research/devecosystem-2018/. Kasutatud: 21.02.2021.

[13] „The State of Developer Ecosystem in 2019,” [Võrgumaterjal]. Saadaval: https://www.jetbrains.com/lp/devecosystem-2019/. Kasutatud: 21.02.2021.

[14] „The State of Developer Ecosystem in 2020,” [Võrgumaterjal]. Saadaval: https://www.jetbrains.com/lp/devecosystem-2020/ . Kasutatud: 21.02.2021.

[15] „CSS Grid Layout Module Level 2,” [Võrgumaterjal]. Saadaval: https://drafts.csswg.org/css-grid/. Kasutatud: 24.02.2021.

[16] A.Kulkarni, „History,” [Võrgumaterjal]. Saadaval: https://wkhtmltopdf.org/status.html. Kasautatud: 24.02.2021.

[17] A.Kulkarni, „Recommendations,” [Võrgumaterjal]. Saadaval: https://wkhtmltopdf.org/status.html. Kasutatud: 24.02.2021. *Recommendations* sektsioon.

[18] „Browser Market Share Worldwide,” [Võrgumaterjal]. Saadaval: https://gs.statcounter.com/browser-market-share. Kasutatud: 24.02.2021.

[19] „About Skia,” [Võrgumaterjal]. Saadaval: https://skia.org/about/. Kasutatud: 06.04.2021.

[20] „Rendering on the Web,” [Võrgumaterjal]. Saadaval: https://developers.google.com/web/updates/2019/02/rendering-on-the-web. Kasutatud: 28.03.2021.

[21] „Most Loved, Dreaded, and Wanted Other Frameworks, Libraries, and Tools,” [Võrgumaterjal]. Saadaval: https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks-wanted2. Kasutatud: 28.03.2021.

[22] „JavaScript,” [Võrgumaterjal]. Saadaval: https://www.jetbrains.com/lp/devecosystem-2020/javascript/. Kasutatud: 28.03.2021.

[23] „Puppeteer API v8.0.0,” [Võrgumaterjal]. Saadaval: https://github.com/puppeteer/puppeteer/blob/v8.0.0/docs/api.md#pagegotourl-options. Kasutatud: 05.04.2021.

[24] „Running Puppeteer in Docker,” [Võrgumaterjal]. Saadaval: https://github.com/puppeteer/puppeteer/blob/main/docs/troubleshooting.md#running-puppeteer-in-docker. Kasutatud: 06.04.2021.

[25] „v5.3.1,” [Võrgumaterjal]. Saadaval: https://github.com/puppeteer/puppeteer/releases/tag/v5.3.1. Kasutatud: 06.04.2021.

[26] „Alpine Linux Chromium package details,” [Võrgumaterjal]. Saadaval: https://pkgs.alpinelinux.org/package/v3.12/community/x86_64/chromium. Kasutatud: 06.04.2021.

[27] „Setting Up Chrome Linux Sandbox,” [Võrgumaterjal]. Saadaval: https://github.com/puppeteer/puppeteer/blob/main/docs/troubleshooting.md#setting-up-chrome-linux-sandbox. Kasutatud: 06.04.2021.

[28] „What is the sandbox?,” [Võrgumaterjal]. Saadaval: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox_faq.md#what-is-the-sandbox. Kasutatud: 06.04.2021.

[29] „Universal tutorial,” [Võrgumaterjal]. Saadaval: https://angular.io/guide/universal#universal-tutorial. Kasutatud: 06.04.2021.

[30] „ng-universal,” [Võrgumaterjal]. Saadaval: https://github.com/nestjs/ng-universal. Kasutatud: 06.04.2021.

[31] „Using absolute URLs for HTTP (data) requests on the server,” [Võrgumaterjal]. Saadaval: https://angular.io/guide/universal#using-absolute-urls-for-http-data-requests-on-the-server. Kasutatud: 06.04.2021.

[32] B. Bishop, „Angular Universal – Relative to Absolute HTTP Interceptor,”, [Võrgumaterjal]. Saadaval: https://bcodes.io/blog/post/angular-universal-relative-to-absolute-http-interceptor. Kasutatud: 08.04.2021.

[33] „Puppeteer docs,” [Võrgumaterjal]. Saadaval: https://pptr.dev/#?product=Puppeteer&version=v5.3.0&show=api-pagegotourl-options. Kasutatud: 07.05.2021.

[34] „Window: load event,” [Võrgumaterjal]. Saadaval: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event. Kasutatud: 07.05.2021

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Martin Sooväli

1 Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "PDF-piletite genereerimise süsteemi kaasajastamine ettevõtte Ridango AS näitel" mille juhendaja on Jaanus Pöial

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

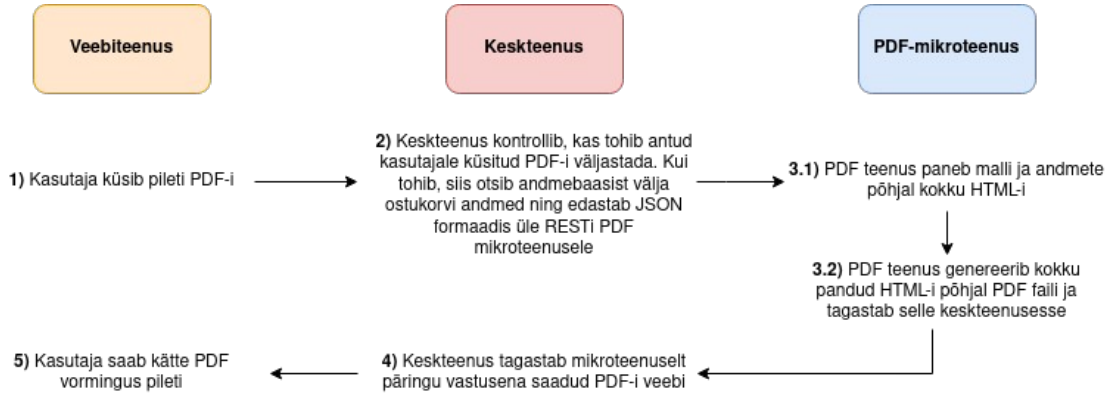
2 Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3 Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

27.04.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Asendatava mikroteenuse struktuuri skeem



Joonis 12: Olemasoleva süsteemi üldvaade

Lisa 3 – Keskteenusest väljuvate andmete struktuur

```
{
  "template": "elron/invoice",
  "lang": "et",
  "json": [
    {
      "product_length": 1,
      .....
      "verification_code": "SDqDMPzx"
    }
  ]
}
```

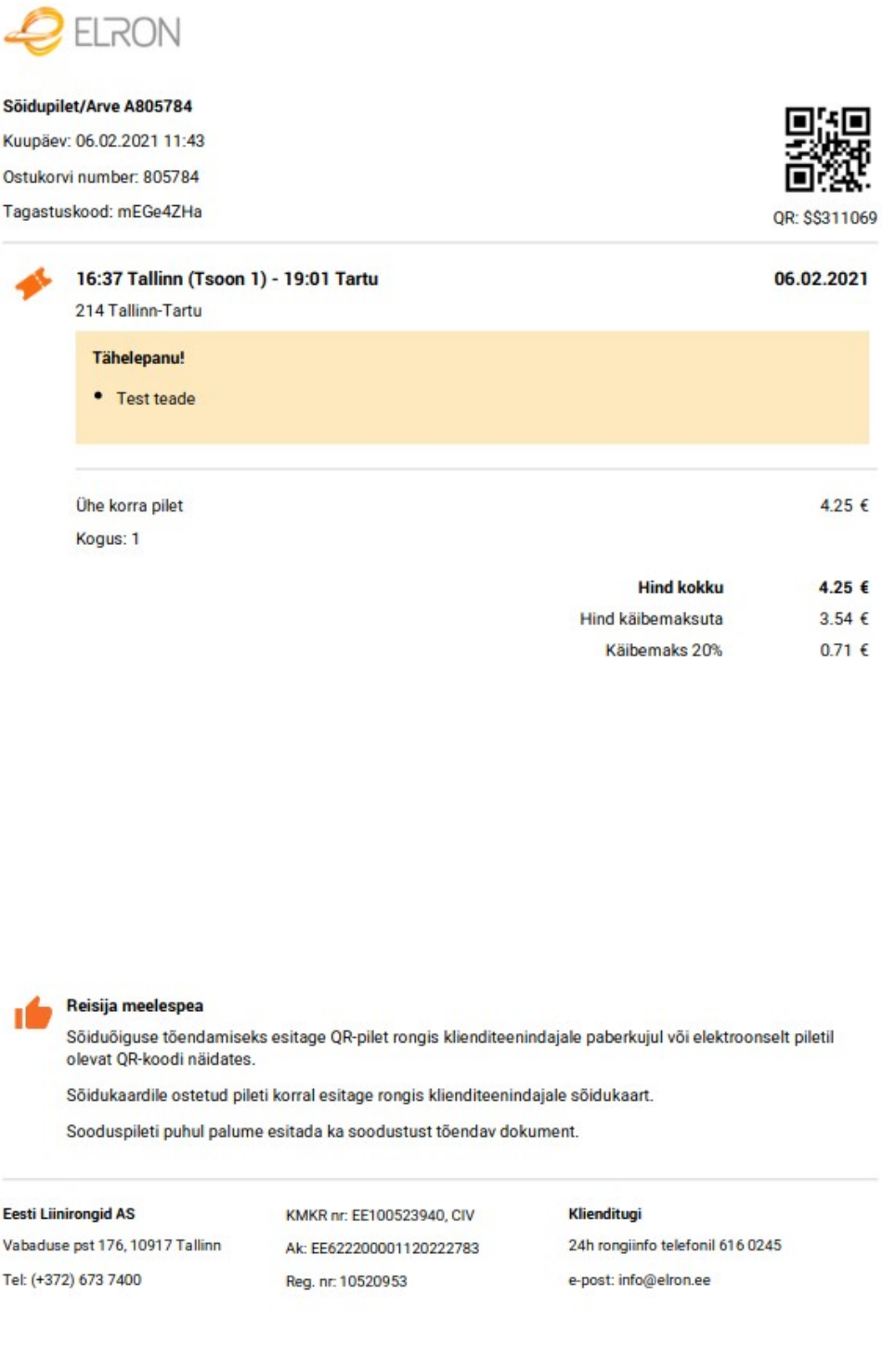
Joonis 13: Üle REST-i liikuvate piletiandmete struktuuri näide

Lisa 4 – V*elocity* malli näide

```
#if( $show_qr )
<div class="table qr-block">
	<div class="row-items">
		<div class="col-item w8">
			<p class="bold">
				Sõidupilet/Arve
				#if ( $data.get(0).shopping_cart_id )
				A$data.get(0).shopping_cart_id
				#{else}
				$!data.get(0).doc_num
				#end
			</p>
			<p>Kuupäev: $date.format("dd.MM.yyyy HH:mm",
$data.get(0).sc_timestamp)</p>
			<p>Ostukorvi number: $!data.get(0).shopping_cart_id</p>
			<p>Tagastuskood: $!data.get(0).verification_code</p>
		</div>
		<div class="col-item w3 float-right right qr">
			<img class="qr-code" id="QRPlaceholder" src="$qr_img"></img>
			<div class="row-items">
				<div class="col-item w12 float-right right qr-code-number">
					QR: $external_code
				</div>
			</div>
		</div>
	</div>
</div>
```

Joonis 14: Velocity malli näide olemasolevast rakendusest

Lisa 5 – Elron AS PDF-pileti näide



ELRON

Sõidupilet/Arve A805784

Kuupäev: 06.02.2021 11:43

Ostukorvi number: 805784

Tagastuskood: mEGe4ZHa

QR: $$311069

16:37 Tallinn (Tsoon 1) - 19:01 Tartu **06.02.2021**

214 Tallinn-Tartu

Tähelepanu!

- Test teade

Ühe korra pilet	4.25 €
Kogus: 1	
Hind kokku	**4.25 €**
Hind käibemaksuta	3.54 €
Käibemaks 20%	0.71 €

Reisija meelespea

Sõiduõiguse tõendamiseks esitage QR-pilet rongis klienditeenindajale paberkujul või elektroonselt piletil olevat QR-koodi näidates.

Sõidukaardile ostetud pileti korral esitage rongis klienditeenindajale sõidukaart.

Sooduspileti puhul palume esitada ka soodustust tõendav dokument.

Eesti Liinirongid AS
Vabaduse pst 176, 10917 Tallinn
Tel: (+372) 673 7400

KMKR nr: EE100523940, CIV
Ak: EE622200001120222783
Reg. nr: 10520953

Klienditugi
24h rongiinfo telefonil 616 0245
e-post: info@elron.ee

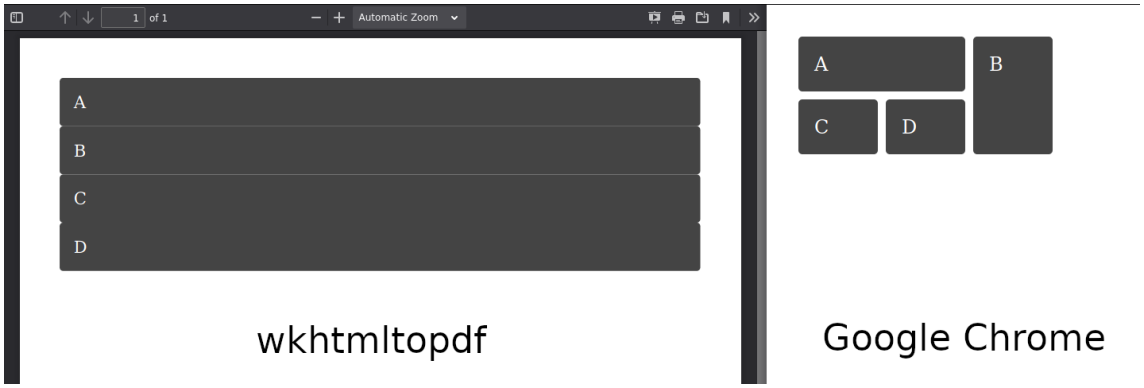
Joonis 15: Elron AS PDF-pilet

Lisa 6 - *wkhtmltopdf* koodinäide

```
package main

import (
	"fmt"
	"log"
	"net/http"

	"github.com/SebastiaanKlippert/go-wkhtmltopdf"
)

func main() {
	// define route for generationg PDF
	http.HandleFunc("/generate-pdf", generatePDF)
	// start listening for requests on port 8080
	_ = http.ListenAndServe(":8080", nil)
}

func generatePDF(w http.ResponseWriter, r *http.Request) {
	// get URL of page which need to be returned in PDF format
	query := r.URL.Query()
	contentUrl := query.Get("content-url")
	fmt.Println(contentUrl)
	// define new pdf generator
	pdfg, err := wkhtmltopdf.NewPDFGenerator()
	if err != nil {
		log.Fatal(err)
	}
	// set page size to A4
	pdfg.PageSize.Set(wkhtmltopdf.PageSizeA4)
	// open page and add to pdf
	page := wkhtmltopdf.NewPage(contentUrl)
	pdfg.AddPage(page)
	// put PDF together
	err = pdfg.Create()
	if err != nil {
		log.Fatal(err)
	}
	// set correct content-type and write created PDF into response
	w.Header().Set("Content-Type", "application/pdf")
	_, _ = w.Write(pdfg.Bytes())
}
```

Joonis 16: wkhtmltopdf koodi näide

Lisa 7 - *wkhtmltopdf* kasutamine CSS grid näitel



Joonis 17: https://gridbyexample.com/examples/code/example5.html katse wkhtmltopdf

Lisa 8 – K6 PDF-generaatori koormustesti kood

```
import http from 'k6/http';
import { check } from 'k6';

export let options = {
   vus: 10,
   duration: '1m',
};

export default () => {
   var url = 'http://localhost:8080/pdf/generate';

   var params = {
      headers: {
         'X-url': 'https://www.google.com/',
      },
   };

   let response = http.get(url, params);
   check(response, {
      'successful response': (resp) => resp.status === 200,
   });
};
```

Joonis 18: K6 koormustesti kood PDF-generaatori vastupidavuse katsetamiseks

Lisa 9 – Teenuste jõudlustestide näidistulemused

```
  execution: local
     script: old-service-perf-test.js
     output: -

  scenarios: (100.00%) 1 scenario, 10 max VUs, 1m30s max duration (incl. graceful stop):
           * default: 10 looping VUs for 1m0s (gracefulStop: 30s)


running (1m00.9s), 00/10 VUs, 511 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  1m0s

     ✓ successful response

     checks.....................: 100.00% ✓ 511  ✗ 0
     data_received..............: 14 MB   229 kB/s
     data_sent..................: 58 kB   948 B/s
     http_req_blocked...........: avg=11.15µs  min=1.75µs   med=5.91µs   max=264.96µs p(90)=12.39µs  p(95)=15.46µs
     http_req_connecting........: avg=926ns    min=0s       med=0s       max=62.1µs   p(90)=0s       p(95)=0s
     http_req_duration..........: avg=1.18s    min=633.58ms med=1.08s    max=3.07s    p(90)=1.64s    p(95)=1.89s
     http_req_receiving.........: avg=235.57µs min=41.37µs  med=175.25µs max=6.71ms   p(90)=396.28µs p(95)=474.81µs
     http_req_sending...........: avg=36.47µs  min=8.63µs   med=30.56µs  max=177.47µs p(90)=67.97µs  p(95)=75.39µs
     http_req_tls_handshaking...: avg=0s       min=0s       med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...........: avg=1.18s    min=633.38ms med=1.08s    max=3.07s    p(90)=1.64s    p(95)=1.89s
     http_reqs..................: 511     8.396625/s
     iteration_duration.........: avg=1.18s    min=633.86ms med=1.08s    max=3.07s    p(90)=1.64s    p(95)=1.89s
     iterations.................: 511     8.396625/s
     vus........................: 10      min=10 max=10
     vus_max....................: 10      min=10 max=10
```

Joonis 19: Ettevõtte Java PDF-teenuse jõudluse tulemus

```
  execution: local
     script: new-service-perf-test.js
     output: -

  scenarios: (100.00%) 1 scenario, 10 max VUs, 1m30s max duration (incl. graceful stop):
           * default: 10 looping VUs for 1m0s (gracefulStop: 30s)


running (1m02.3s), 00/10 VUs, 292 complete and 0 interrupted iterations
default ✓ [======================================] 10 VUs  1m0s

     ✓ successful response

     checks.....................: 100.00% ✓ 292  ✗ 0
     data_received..............: 7.3 MB  117 kB/s
     data_sent..................: 33 kB   529 B/s
     http_req_blocked...........: avg=11.11µs  min=1.86µs  med=4.16µs   max=342.39µs p(90)=5.44µs   p(95)=6.83µs
     http_req_connecting........: avg=2.64µs   min=0s      med=0s       max=187.37µs p(90)=0s       p(95)=0s
     http_req_duration..........: avg=2.1s     min=1.39s   med=1.78s    max=8.18s    p(90)=2.94s    p(95)=3.28s
     http_req_receiving.........: avg=145.45µs min=53.54µs med=110.13µs max=2.7ms    p(90)=169.49µs p(95)=222.09µs
     http_req_sending...........: avg=23.99µs  min=10.38µs med=19.78µs  max=350.07µs p(90)=33.94µs  p(95)=45.15µs
     http_req_tls_handshaking...: avg=0s       min=0s      med=0s       max=0s       p(90)=0s       p(95)=0s
     http_req_waiting...........: avg=2.1s     min=1.39s   med=1.78s    max=8.18s    p(90)=2.94s    p(95)=3.28s
     http_reqs..................: 292     4.683786/s
     iteration_duration.........: avg=2.1s     min=1.39s   med=1.78s    max=8.18s    p(90)=2.94s    p(95)=3.28s
     iterations.................: 292     4.683786/s
     vus........................: 3       min=3  max=10
     vus_max....................: 10      min=10 max=10
```

Joonis 20: Valminud PDF-teenuse jõudluse tulemus