TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Kristian Alex Laherand 206264IADB

Ilusalongi veebirakenduse prototüüp

Bakalaureusetöö

Juhendaja: Meelis Antoi

MSc

Tallinn 2023

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: Kristian Alex Laherand

14.05.2023

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua veebirakendus, mille otstarbeks on anda infot Ilusära kosmeetikud ilusalongi kohta. Töö tulemus pakub huvi kõikidele inimestele, kes otsivad ilusalongi, kuhu iluprotseduure tegema minna. Töö tulemus oleks kasulikuks infoallikaks antud ilusalongi potentsiaalsetele klientidele, andes ülevaate salongis tehtavatest protseduuridest ja nende hindadest; samuti salongi töötajatele, andes võimaluse edastada potentsiaalsele klientuurile informatsiooni pakutavatest teenustest; ja ka teiste ilusalongide omanikele, võimaldades neil kursis olla konkurentide hinna- ja kliendipoliitikast.

Bakalaureusetöös analüüsitakse eksisteerivaid ilusalongi koduleheküle lahendusi ja defineeritakse ilusalongi veebirakenduse eeldused ja nõuded. Seejärel seletatakse lahti veebirakenduse arhitektuur koos kasutatud tehnoloogiate ja arenduse protsessiga.

Töö tulemuseks on töötav ja intuitiivselt arusaadav ilusalongi veebirakendus, kus on näidatakse infot ilusalongi kohta (asukoht, parkimine, sissepääs jne) ning tutvustatakse salongis töötavaid inimesi ja nende poolt pakutavaid teenuseid.

Lõputöö on kirjutatud eesti keeles ja sisaldab 43 lehekülge teksti, 6 peatükki ja 10 joonist.

Abstract

Beauty Salon Web Application Prototype

The aim of the current thesis is to create a web application that provides information about the Ilusära cosmetics beauty salon. The results of this work are of interest to everyone looking for a beauty salon to go to for beauty treatments. The work is of interest to potential clients of the salon, providing an overview of the procedures performed in the salon and their prices; also, to the salon's employees, allowing them to provide potential clients with information about the offered services; and also to the owners of other beauty salons, enabling them to stay informed about competitors' pricing and customer policies.

The bachelor's thesis analyzes existing beauty salon website solutions and defines the prerequisites and requirements for a beauty salon web application. Subsequently, the web application architecture, the technologies used, and the development process are explained.

The result of the work is a functioning and intuitively understandable beauty salon web application that displays information about the salon (location, parking, entrance, etc.) and introduces the people working in the salon and the services they offer.

The thesis is written in Estonian and contains 43 pages of text, 6 chapters, and 10 drawings.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* – funktsioonide ja protseduuride kogum, mis võimaldab luua rakendusi, mis pääsevad ligi rakenduse funktsioonidele või andmetele
Backend	Serveripoolne keskkond
BLL	*Business Logic Layer* – kiht, kus töödeldakse andmeid ning määratakse neile ligipääsetavuse õigused
Casting	Andmete muutmine ühest andmetüübist teiseks
CMS	*Content Management System* – sisuhaldustarkvara
CORS	*Cross-Origin Resource Sharing* – HTTP päisel põhinev mehhanism, mis võimaldab serveril näidata ressursse, mis ei ole tema enda omad
CRUD	Akronüüm, mis tähendab 4 funktsiooni: *create, read, update, delete* ehk loomine, lugemine, uuendamine ja kustutamine
CSS	*Cascading Style Sheets* – märgistuskeel, millega küljendatakse veebilehte
DAL	*Data Access Layer* – kiht, kust saadakse andmeid kätte
DevOps	*Software development (Dev) + IT operations (Ops)* – metoodika, mida kasutatakse tarkvara arendamisel. Seda kasutatakse juhtpraktikate ja tööriistade komplektina
DOM	*Document Object Model* – liides, mis käsitleb HTML- või XML-dokumenti puustruktuurina, milles iga sõlm on objekt, mis esindab dokumendi osa
DTO	*Data Transfer Object* – andmeedastusobjekt

Frontend	Kasutajaliides, kliendipoolne keskkond
HTML	*Hypertext Markup Language* – standardiseeritud süsteem tekstifailide redigeerimiseks, võimaldades muuta fondi-, värvi-, graafikaefekte
HTTP/HTTPS	*Hypertext Transfer Protocol* – hajutatud edastusprotokoll, mis võimaldab kasutajatel veebis andmeid edastada. HTTPS on HTTP, mis on krüpteerimisega ja kontrollimisega
IDE	*Integrated development environment* - Integreeritud arenduskeskkond
JSON	*JavaScript Object Notation* – faili- või andmevahetusvorming, mis kasutab atribuudi-väärtuse paaridest koosnevat andmeobjekti salvestamiseks või edastamiseks
JSX	JavaScript XML – märgistus süntaks, mis lubab kirjutada HTML-i otse Reacti
REST	*Representational state transfer* – tarkvara arhitektuuristiil, mis jälgib kindlaid piiranguid
URL	*Uniform Resource Locators* – unikaalne nimi, mis määratakse veebiaadressitele
UX	*User experience* – kasutajakogemus
XML	*Extensible Markup Language* – märgistuskeel, mis sätib reeglid mistahes andmete määratlemiseks
YAML	*Yet Another Markup Language* – inimloetav andmete serialiseerimise keel

SISUKORD

Jooniste loetelu

1 SISSEJUHATUS

Iga inimene soovib viisakana ja professionaalsena välja näha, ja selleks tuleb enda eest hoolt kanda. Selle jaoks tuleb ennast hooldada kvaliteetsete ilutoodetega ning soovitatav on käia ilusalongis protseduuridel. Üks edukas iluprotseduure pakkuv salong peaks pakkuma laia valikut tooteid, mis vastavad klientide soovidele ja vajadustele ja teenuseid kõrgeima kvaliteediga. Et inimesed tahaksid alati tagasi tulla ja tunneksid ennast salongis mugavalt, tuleb neile pakkuda meisterlikku teenindust, kvaliteetseid tooteid ning anda klientidele võimalust tutvuda teenuste ja toodete valikuga kergelt mõistetava kaasaegse veebirakendusega.

Antud lõputöö käsitleb ilusalongi veebirakendust Eesti kontekstis. Töö kirjutamise käigus uuritakse teisi Eestis töötavaid kosmeetikasalongide veebikeskkondi.

Veebirakenduse tellija töötab ilusalongis kosmeetikuna ning paremaks turustamiseks, info edasiandmiseks, samuti selleks et olemasolevad ja potentsiaalsed kliendid saaksid teenuste ja toodete hindadega tutvuda, tagasisidet jätta, pilte vaadata, töötajatega tutvuda ning logisid jälgida, vajab antud ilusalong veebirakendust.

1.1 Metoodika

Antud lõputöö raames lahendatakse olemasolevat probleemi, uurides selleks juba eksisteerivaid lahendusi ja nende eeliseid ja puudusi. Lõputöös pakutakse välja sobiv infotehnoloogiline lahendus, mille fookuses on kasutusmugavus ja mida saaks tulevikus edasi arendada.

Kasutajate vajadused on esitatud kasutajalugude kujul, mis on grupeeritud funktsionaalseteks ja mittefunktsionaalseteks nõueteks. Tehniliste lahenduste valimisel on põhjendatud serveripoolsete ja kliendipoolsete tehnoloogiate valikud.

Lahenduse valmimist on kirjeldatud erinevate osadena, mis hõlmavad rakenduse kasutajaliidese, andmebaasi ning serveri- ja kliendipoolse lahenduse disaini. Diplomitöös on kirjeldatud ka lahenduse testimist ning võimalikke edasiarendusi, mis ei jää diplomitöö skoopi.

2 Ülevaade probleemist

Veebirakendus on üks parimaid viise olla läbipaistev ning näidata inimestele, millega üks ettevõte või kollektiiv tegeleb. Ilma veebis enda info näitamiseta peavad töötajad telefoni teel tooteid, teenuseid ja nende hinnakirju tutvustama. Samuti peab ilusalong tegema palju reklaami, et endale uusi kliente meelitada. Et kirjeldatud olukordi vältida, on üks mõistlikumaid variante seda infot veebis näidata. Selleks on võimalus endale näiteks Facebooki või mõne muu sotsiaalmeedia lehekülg teha, kuid läbi sotsiaalmeedia on väga raske ilusalongi „aurat“ tutvustada. Sellepärast eelistatakse sageli oma infot näidata veebilehel, tehes sellele meeleolu edasiandmiseks vastav disain ning kasutajakogemus.

2.1 Eksisteerivad lahendused

Ilusalongil pole hetkel kohta veebis, kus näeb nende kohta käivat infot. Kliendid peavad info saamiseks kas töötajatele helistama või siis salongi ise füüsiliselt kohale tulema. Ainuke eksisteeriv informatsioon antud kosmeetikasalongi kohta on nende nime guugeldamisel antavad telefoninumber, asukoht ja tööaeg.

2.1.1 Staatiline lahendus

Staatiline lahendus sisaldab navigatsioonipaneeli, kus on erinevate lehtede lingid. Lehed on disainitud olema dünaamilised, et rohkem infot korraga ekraanile mahuks. Näiteks tehti teenuse tüübi järgi järjend, mille peale vajutades avaneb teine järjend, kus näidataks kõiki selle tüübi alla käivaid teenuseid koos hindadega. Kõik veebilehe lehed on järgmised: avaleht, kontaktid, pildid, tooted, hinded, teenused, allahindlused ja kampaaniad.

2.2 Uue lahenduse skoop

Arvestades diplomitöö mahuga, on diplomitöö autor määranud skoobiks rakenduse MVP (*Minimum Viable Product*), mis on rakenduse minimaalne elujõuline toode. Selleks peab klient suhtlema veebiteenusega ning suutma näidata kõiki veebilehe lehti ning rippmenüüd, kust saab rakenduses navigeerida. Samuti peab valmis olema administraator kasutaja paneel,

kust töötajad administraatori õigustega saavad veebilehele informatsiooni lisada, seda uuendada või kustutada.

3 Loodava veebirakenduse analüüs

Veebirakenduse arendamine on protsess, mis nõuab süstemaatilist lähenemist ja põhjalikku tehnoloogiliste valikute analüüsi. Antud peatükk keskendub esitatud ärinõuetele, diplomitöö käigus loodavale veebirakenduse arendusprotsessile ning tehnoloogilistele valikutele, mis on tehtud, et tagada veebirakenduse kvaliteet, paindlikkus ja kasutajasõbralikkus. Käsitletakse valitud programmeerimiskeeli, arenduskeskkondi, andmebaasi ja serveri lahendusi ning peamisi funktsionaalsusi ja tuleviku väljavaateid, mis võimaldavad rakendusel edasi areneda.

3.1 Nõuete määramine

Nõuete määramisel konsulteeriti ilusalongi töötajatega, et teada saada, mis on nende ootused antud rakendusest ja milline võiks olla esialgne disain. Disainist räägiti rohkem veebilehele pandavatest komponentidest ning nende paigutusest, kui ülejäänud rakenduse välimusest.

3.1.1 Funktsionaalsed nõuded

Tavakasutaja jaoks vajalikud funktsionaalsed nõuded:

- Tavakasutajana soovin registreeruda kasutajaks läbi veebirakenduse keskkonna
- Tavakasutajana soovin sisse ja välja logida
- Tavakasutajana soovin näha ilusalongi poolt pakutavaid teenuseid ja nende hindasid
- Tavakasutajana soovin näha ilusalongi poolt pakutavaid tooteid ja nende hindasid
- Tavakasutajana soovin jätta tagasisidet ilusalongi külastamise kogemuse kohta
- Tavakasutajana soovin lugeda teiste klientide tagasisidet ilusalongi külastamise kogemuse kohta
- Tavakasutajana soovin näha ilusalongi töötajaid

Töötaja jaoks vajalikud funktsionaalsed nõuded:

- Töötajana soovin lisada, muuta ja eemaldada hinnakirjas olevaid tooteid
- Töötajana soovin lisada, muuta ja eemaldada hinnakirjas olevaid teenuseid
- Töötajana soovin näha logi külastajate kohta
- Töötajana soovin lugeda klientide tagasisidet ilusalongi külastamise kogemuse kohta

3.1.2 Mittefunktsionaalsed nõuded

Tavakasutaja jaoks vajalikud mittefunktsionaalsed nõuded:

- Tavakasutajana soovin, et veebirakendus oleks kasutajasõbralik, see tähendab arusaadava navigatsiooni, kujunduse ja komponentide paigutusega
- Tavakasutajana soovin, et minu andmetele oleks tagatud privaatsus ja turvalisus
- Tavakasutajana soovin, et veebilehte saaks kasutada kõikides levinumates veebilehitsejates
- Tavakasutajana soovin, et veebirakendus oleks stabiilne ja ilma pikemate viivitusteta

Administraatori jaoks vajalikud mittefunktsionaalsed nõuded:

- Administraatorina soovin, et kogu kood asub tsentraalses repositooriumis
- Administraatorina soovin, et kogu veebirakenduse kood on mugavalt hallatav
- Administraatorina soovin, et veebirakendus on skaleeruv

3.1.3 Tulevikus loodavad funktsionaalsused

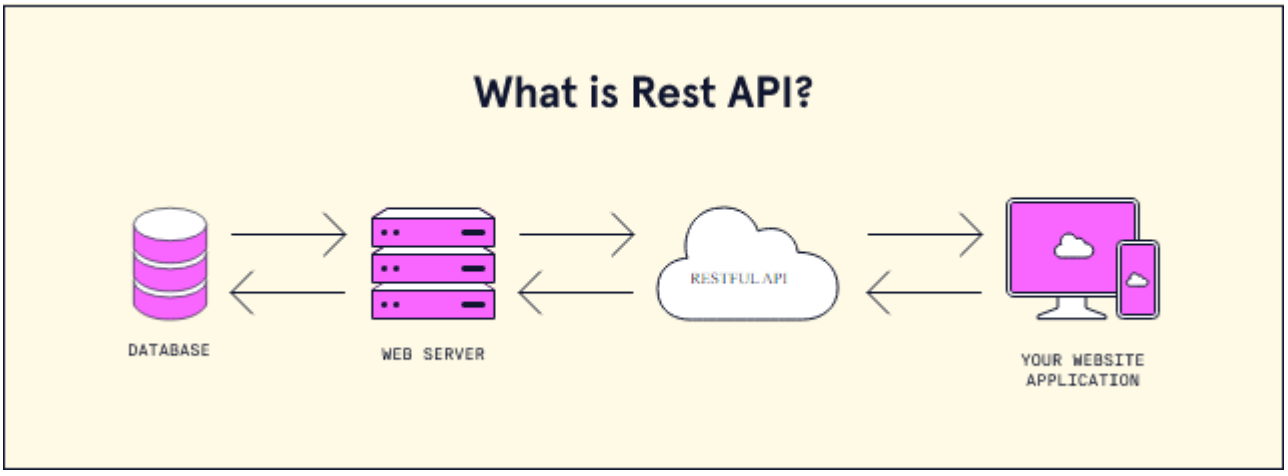
Kõige tähtsamaks tulevikus loodavaks funktsionaalsuseks on lisada rakendusele keelevahetuse funktsionaalsus. Tähtis on ka testida nii serveri poolt rakendusest kui ka klientrakendust nii ühik kui ka integratsioonitestidega. Tulevikus saab vajadusel lisada rakendusele broneerimissüsteemi, kus klientidel on võimalik endale sobival kuupäeval ja kellaajal teenust broneerida, juhul kui mõni töötajatest hakkab kindla graafiku alusel töötama. Suuremaks tulevikus loodava võimalikuks funktsionaalsuseks on ka e-poe tegemine, siis saab ilusalong teenida lisa tulu, kui inimesed soovivad endale mingit toodet soetada. Selleks tuleks ka kokkulepped teha kulleriettevõtetega, kes tooted klientide juurde turvaliselt toovad. Lisada saab ka läbi sotsiaalmeedia registreerimine, siis saab vaid ühe hiire vajutusega kõik vajalikud kliendi andmed kätte. Kui ilusalong läheb väga populaarseks, siis ei ole välistatud mobiilirakenduse loomine.

3.2 Tehnoloogia valik

Veebirakenduse arendamisel sõltub tehnoloogiate valik rakendusele vajalikust funktsionaalsusest ning arendaja kogemustest ja isiklikest eelistustest. Antud lõputöö raames

otsustati valida vaid levinumatest arenduskeelte seast, millel on korralik dokumentatsioon ning palju kasutajaid. Sellest valikust eraldati CMS (*Content Management System*) lahendused, sellepärast et nendega ei ole alati võimalik saavutada vajalikku funktsionaalsust, neid on testida ja siluda raskem, kui mõni funktsioon või komponent ei tööta nii nagu peaks ning need on loodud kergete veebilehtede arendamise jaoks.

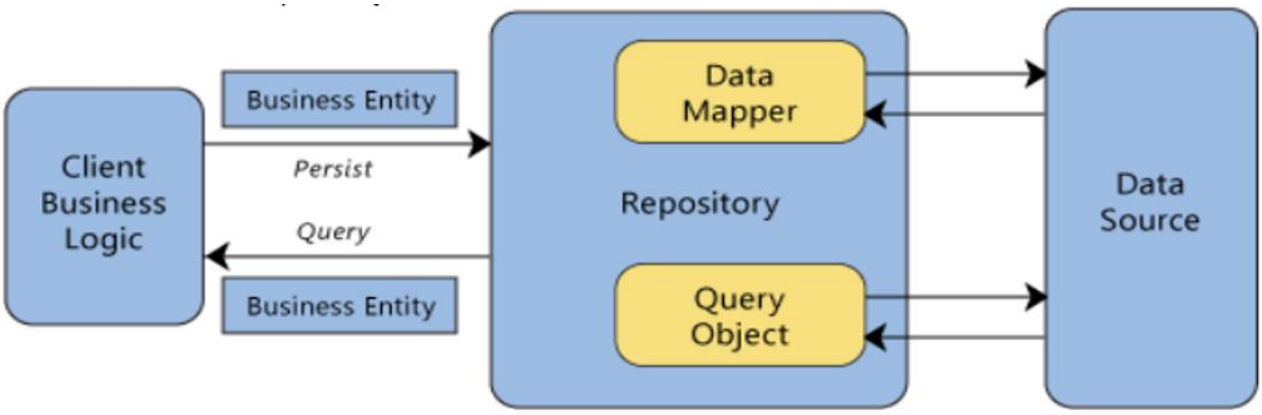
Rakenduse arendamisel on planeeritud kasutada erinevaid mustreid, mis kihistavad ning aitavad arenduse ajal koodi disainida [1]. Serveripoolsel rakenduse osal kasutatakse REST (*Representational State Transfer*) lähenemist, mis tagab veebis süsteemide koostoimimise. Tegemist on lähenemisega, kus serveripoolne ega kliendipoolne rakenduse osa ei pea teadma, mis olekus parasjagu teine on [2] (Joonis 1).



Joonis 1. REST API arhitektuur [3].

REST lubab struktureerida andmeid eri formaatides nagu näiteks tava tekst, HTML, XML, YAML ja JSON. Nendest kõige levinum on tänapäeval JSON, sest selle süntaks on väga otsekohene ning teeb andmete muutmisel või loomisel tekkinud vead lihtsasti kättesaadavaks.

Et kood oleks lihtsasti testitav ning edasi arendatav serveripoolel kasutatakse koodi disainimisel hoidlate mustrit, mis eraldab koodi juppideks ning kergemalt hallatavaks [4] (Joonis 2).



Joonis 2. Hoidlate muster [5].

Hoidlate muster eraldab DAL-i (*Data Access Layer*), mis töötab domeeni olemitega ja teostab andmetele juurdepääsu loogikat ja kaardistab selle BLL-i (*Business Logic Layer*) olemiteks, kus andmetele rakendatakse äriloogikat.

Kõikidele kihtidele tehakse ka liidesklassid, mis määravad ära reeglid, mida antud kihi klass peab jälgima. Liidestes kasutatakse geneerilisi tüüpe, mis annavad võimaluse kasutada mingit klassi või meetodit erinevates kohtades erinevate tüüpi andmetega [6]. See välistab olemeid nii nimetatud *cast*'imisest teiseks andmetüübiks ehk olemite muutmist teiseks andmetüübiks ilma et kompilaator teaks kas seda saab teha. See aitab programmeerija vigu vältida ja teeb veakoodid arusaadavaks. Lisaks *cast*'imine on väga ressursse nõudev operatsioon ning muudab programmi aeglasemaks.

3.2.1 Serveripoolse programmeerimiskeele valik

Tänapäeval on välja töötatud tuhandeid programmeerimiskeeli, kuid ainult vähesed neist leiavad laialdast kasutust [7]. Iga programmeerimiskeel on mõeldud millegi jaoks, näiteks on keeli operatsiooni süsteemide kirjutamiseks, veebilehtede loomiseks, teadusliku tarkvara loomise jaoks. Samuti mõned programmeerimiskeeled on kindla otstarbega, näiteks arvutimängude loomiseks või mõne operatsioonisüsteemi töölaua rakenduse tarkvara kirjutamiseks.

Sellegipoolest on programmeerimiskeeli, mida saab kasutada peaaegu igas rakenduses. Kõige suurema kasutajaskonnaga serveripoolsed keeled on tänapäeval objektorienteeritud programmeerimise keeled ja nendest tuntuimad on:

- Python – tüüpimata programmeerimiskeel, mida kasutatakse veebilehtede ja tarkvara loomiseks, ülesannete automatiseerimiseks ja andmete analüüsimiseks. Python ei ole spetsialiseerunud ühelegi konkreetsele probleemile ehk ta on üldotstarbeline [8]
- JavaScript (TypeScript) – skriptimiskeel, mis võimaldab luua dünaamiliselt värskendatavat sisu, juhtida multimeediat, animeerida pilte jne [9]. Seda saab kasutada nii rakenduse serveripoolse osa kui ka kliendipoolse osa arendamiseks.
- Java – platvormist sõltumatu objektorienteeritud tüübitud programmeerimiskeel. Java on üks populaarsemaid programmeerimiskeeli maailmas [10].
- C# - moderne objektorienteeritud tüübitud programmeerimiskeel. C# võimaldab arendajatel luua erinevaid turvalisi ja töökindlaid rakendusi, mis töötavad .NET-is [11].
- Ruby – programmeerimiskeel, mis on kõige kuulsam selle kasutamise poolest veebiarendustes, kuid kasutatakse ka automatiseerimises, käsurea tööriistade arendamisel, staatiliste veebilehtede genereerimisel, DevOps (*Software development (Dev) + IT operations (Ops)*), andmekoorimisel veebis ja andmetöötluses. See on väga mitmekülgne ja mitmeotstarbeline [12].

Arvestades sellega, et uue programmeerimiskeele õppimine on väga aeganõudev ning antud diplomitöö autoril on loetelus nimetatud programmeerimiskeeltest arvestatavat kogemust ainult C# ja Java keeltega, siis on mõistlik diplomitöö mahu raames valida arenduseks juba autorile tuttav programmeerimiskeel. Kuna C# on kiirem kui Java, kood kompileeritakse otse binaarseks koodiks, C#-l on võimsam tüübisüsteem ning lihtsam süntaks kui Javal, on lõputöö autor otsustanud valida serveripoolseks programmeerimiskeeleks C#-i.

3.2.2 Serveripoolsed raamistikud

Tulenevalt eelmises lõigus kirjutatust, on serveripoolel arendamiseks planeeritud kastutada programmeerimiskeelt C#. Sellest järeldub, et diplomitöös kasutatakse .NET Framework raamistikku. .NET-süsteem sisaldab tööriistu, teeke ja keeli, mis toetavad kaasaegset skaleeruvat ja suure jõudlusega tarkvaraarendust. .NET-i hooldab ja toetab aktiivne arendajate kogukond [13].

3.2.3 Kliendipoolne tehnoloogia

Kliendipoolsest tehnoloogiast rääkides, eriti veebilehtedest, ei saa üle ega ümber JavaScriptist. JavaScriptiga saab luua dünaamilisi ja interaktiivseid lahendusi veebis. Tänapäeval enamus funktsioone ja rakendusi, mis on modernses veebis hädavajalikud, on kodeeritud mingil kujul just JavaScriptis [14].

Järgnevalt on väljatoodud ja lühidalt kirjeldatud populaarsemad JavaScripti raamistikud:

- React – Facebooki poolt välja töötatud raamistik, mis on mõeldud suurte ja dünaamiliste veebirakenduste loomiseks. Reacti eelis on tema lihtsus ja tõhusus.
- Angular – Google'i poolt välja töötatud raamistik, mis on väga struktureeritud. See teeb ta ideaalseks valikuks keeruliste rakenduste arendamisel.
- Vue.js – Lihtne ja paindlik raamistik, mis on avatud lähtekoodiga. Sellega saab interaktiivseid veebirakendusi luua.

Kuna kõik nimetatud raamistikud on vabavaralised lahendused, siis rahastamine ei ole nende kasutuse puhul takistuseks. React on efektiivsem DOM-i (*Document Object Model*) mõjutamisel kui Angular või Vue.js. Angular on aeglane, kui rakendus on suur. Diplomitöö autoril on kogemust vaid Vue.js-i ja Reactiga ning kuna React on populaarsem, hästi dokumenteeritud ja kergelt hallatav, siis otsustati kliendipoolseks tehnoloogiaks valida React [15].

3.3 Andmebaasi valik

Andmebaasi valiku aluseks on see kas kasutada SQL andmebaasi või mõnda teist andmebaasikeele andmebaasi. SQL andmebaasid edestavad teisi andmebaase oma tõhususe, andmete kerge haldamise, turvalisuse ja võimekuse poolest. Need on kõige populaarsemad andmebaasid maailmas ning neid toetavad laialdaselt kasutust leidvad platvormid ja rakendused. Sellest järeldub, et SQL andmebaasid on hästi dokumenteeritud ja paljude arendajate poolt kasutatud, mis tähendab, et nad saavad rohkem toetust ja ressursse kui teised [16].

Siin on mõned populaarsemad SQL andmebaasid ja nende kirjeldused:

- MySQL - avatud lähtekoodiga andmebaas, mis on laialdaselt kasutusel veebirakendustes. See on kiire, tõhus ja turvaline ning pakub laia valikut funktsioone ja indekseerimist. [17]
- PostgreSQL – võimas, avatud lähtekoodiga objekt-relatsiooniline andmebaas. Andmebaas pakub kaasaegseid funktsioone nagu näiteks JSON tugi [18].
- SQLite – teek, mis rakendab iseseisvat serverita, SQL-andmebaasi mootorit. See on nullkonfiguratsiooniga andmebaas, mis tähendab, et sarnaselt teiste andmebaasidega ei pea seda süsteemis konfigureerima [19].
- Microsoft SQL Server - on suuremahuliste infotehnoloogia keskkondade jaoks mõeldud relatsiooniline andmebaasihaldussüsteem, mis toetab erinevaid tehingutöötlus-, ärilise intelligentsuse- ja analüüsirakendusi. [20].
- Oracle Database - üks populaarsemaid SQL andmebaase maailmas. See on suurte ettevõtete jaoks kavandatud ning pole vabavaraline ehk see maksab.

Nimetatud andmebaasidest on vabavaralised vaid PostgreSQL, MySQL ja SQLite. PostgreSQL pakub nimetatud andmebaasidest kõige suuremat hulka toetatud andmetüüpe, indekseid ning sellega on ka diplomitöö autoril kogemust, on otsustatud andmebaasiks valida just PostgreSQL.

3.4 Arenduskeskkonna valik

Veebirakenduse serveri osa arendamisel kasutatava arenduskeskkonna valikutes olid JetBrains Rider ja Microsoft Visual Studio. Mõlemad keskkonnad on IDE-d ehk integreeritud arenduskeskkonnad, mida kasutatakse erinevates programmeerimiskeeltes ja erinevatel platvormidel tarkvara arendamiseks.

Järgnevalt on välja toodud peamised erinevused antud platvormide vahel:

- Platvormid: Rider on JetBrainsi poolt loodud, mis on saadaval ja kasutatav Windowsi, macOS ja Linuxi platvormidel, samas kui Visual Studio on Microsofti arenduskeskkond, mis on mõeldud Windowsi platvormile.
- Keeled: Visual Studio on mõeldud peamiselt Microsofti platvormidel töötavate tarkvaralahenduste arendamiseks, nagu C# ja Visual Basic. Rider on aga mõeldud

laiemaks kasutamiseks ja see toetab paljusid erinevaid programmeerimiskeeli nagu näiteks C#, Java, Kotlin, Ruby ja JavaScript.

- Hinnakujundus: Rider on tasuline tarkvara, millel on tasuta prooviversioon, samas kui Visual Studio pakub tasuta ja tasulist versiooni, sõltuvalt programmeerija vajadustest.
- Funktsioonid: Mõlemad pakuvad laia valikut funktsioone ja tööriistu tarkvaraarenduseks. Visual Studio edestab Riderit oma integreeritud tarkvaraplatvormi ja pistikprogrammide valiku poolest. Rider aga edestab Visual Studiot arendusprotsessi sujuvuses, kasutajakogemuses ja efektiivsuses tänu kiiremale käivitamisele ja jõudlusele, tööriistadele ja refaktoreerimisele [21].

Võib öelda et mõlemad arenduskeskkonnad on väga kõrgel tasemel ning pidevas arenduses. Kuna diplomitöö autoril on rohkem kogemust Rideriga ning veebirakenduse arendamisel tehakse tööd mitmel operatsioonisüsteemil, siis mõistlikumaks valikuks on antud juhul Rideri keskkond.

Veebirakenduse kliendipoolse osa arendamisel kasutatava arenduskeskkonna valikutes olid Microsoft Visual Studio Code ja JetBrains WebStorm. Diplomitöö autoril on kogemust vaid Microsoft Visual Studio Code-ga ning kuna seda on kerge kasutada ja ta on kergekaalulisem kui WebStorm [22]. Sellest tulenevalt on otsustatud valida kliendipoolse osa arenduskeskkonnaks Microsoft Visual Studio Code [1]

3.5 Veebirakenduse haldus

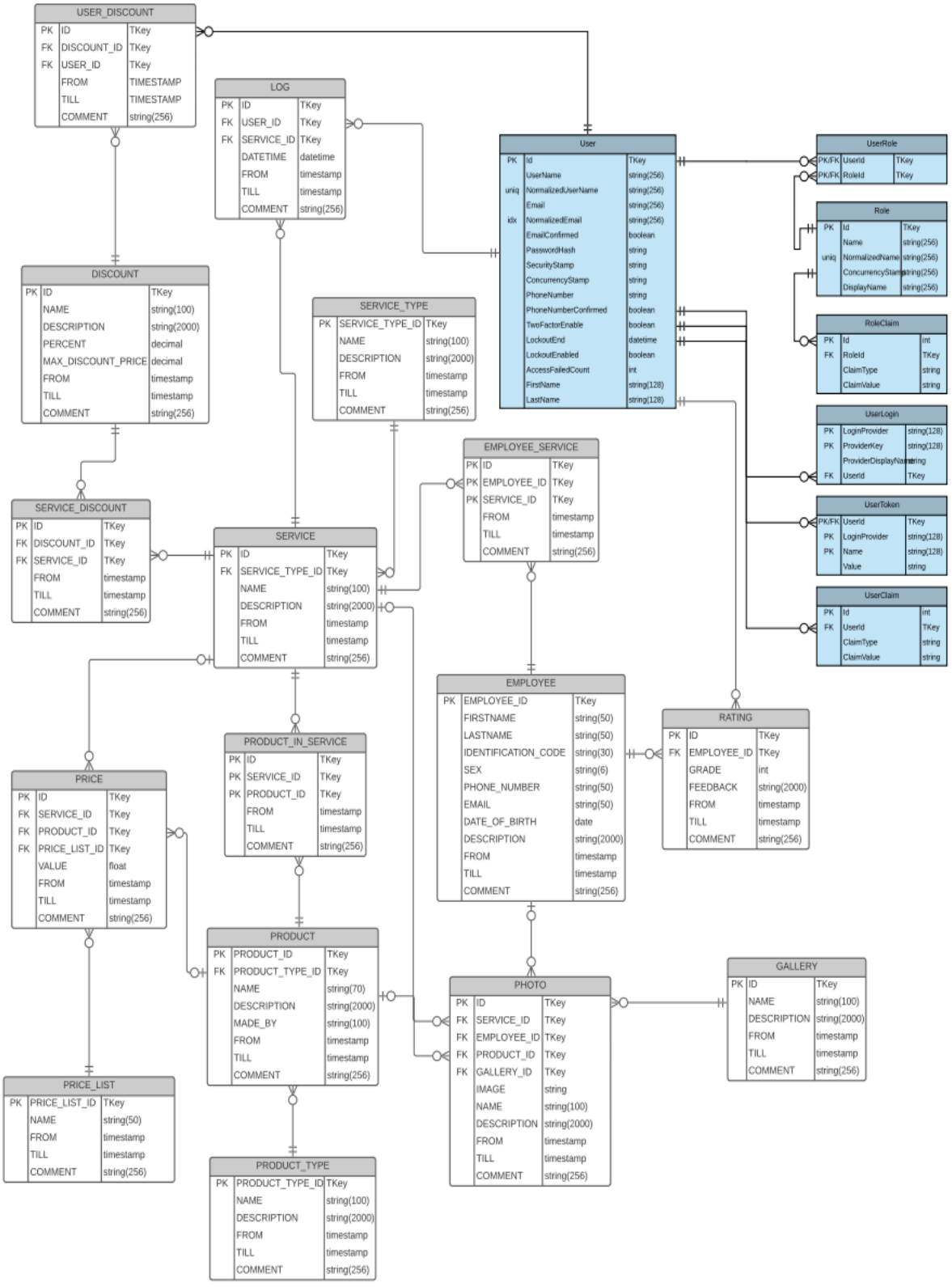
Veebirakenduse haldamine on oluline veebirakenduse tõrgeteta toimimiseks ning hea kasutajakogemuse tagamiseks. Selleks peab tagatud olema *hosting* ehk veebirakenduse veebis kättesaadavaks tegemine, turvalisus, et andmeid ei saaks varastada, pidevad juurde arendused ning vigade parandamine, et võimalike turvaauke oleks vähem ning et rakendus sujuvamaks muuta, andmete varundamine, juhuks kui need hävivad või tekib mingi viga, mis muudab andmed vigaseks ja veebirakenduse jõudluse jälgimine, et näha võimalikke probleeme nagu suur andmemaht.

Veebirakenduse serveerimiseks on vaja serverilahendust ning tänapäeval on neid kaks: füüsiline server ehk server pannakse üles füüsilise riistvara peale eraldades sellele mälu või pilv ehk serverid, mis ei ole kindla riistvara peal, vaid virtuaalses keskkonnas. Füüsilise serveri eelis on tõhusus ja kiire töötlemine, kuna neil on hea jõudlus. Virtuaalse serveri eeliseks on paindlik serveri mahu suurendamine või vähendamine, kergem serveri enda haldamine, kergem turvalisuse haldamine, kergem mastaapsuse muutmine ja on pikema perioodi vältel soodsamad. Üheks suureks virtuaalse serveri eeliseks on töökoormuse teisaldamine üle virtuaalse keskkonna [23].

Kuna diplomitöös arendataval veebirakendusel ei ole tarvis ülemäära palju jõudlust, on otsustatud kasutada haldamisel virtuaalset serverit.

3.6 Andmebaasi disain

Kasutajalugudest lähtuvalt tekkis hea pilt sellest, milline andmebaasi olemi-suhte diagramm peaks välja nägema. Olemi-suhte diagramm oli loodud kasutades rakendust Lucidchart ning diagrammile on lisatud ka migratsiooni käigus programmi enda poolt loodavad kasutaja tabelid. Joonis 3 näitab antud diplomitöös kasutatud olemi-suhte diagrammi.



Joonis 3. Olemi-suhte diagramm.

3.7 Analüüsi kokkuvõte

Veebirakenduse analüüs keskendub nõuete määramisele, tehnoloogia valikule ja tulevastele funktsionaalsustele. Kõik valikud tehti arvestades võimalikult madalat eelarvet, kuid samas võimalikult tõhusat ja kiiret edasiliikumist. Arendamiseks valiti paindlikud programmeerimistehnikad ja disainimeetmed, et rakendust oleks tulevikus lihtne edasi arendada ja uusi funktsionaalsusi lisada. Samuti sooviti teha palju liideseid, et kogu rakendus muuta kergemalt loetavamaks ning kood oleks uuesti kasutatav. Ilusalongi töötajatega konsulteerimise käigus selgitati välja äri- ja süsteeminõuded.

Tulevikus võib rakendusele lisada broneerimissüsteemi, e-poe, läbi sotsiaalmeedia registreerimise, mobiilirakenduse ja keelevahetuse funktsionaalsused. Tehnoloogia valikus keskenduti levinumatele arenduskeeltele, millel on hea dokumentatsioon ja palju kasutajaid. Valitud keeled olid C# ja Java, kuid C# valiti serveripoolse programmeerimiskeelena, kuna see on kiirem ja lihtsama süntaksiga.

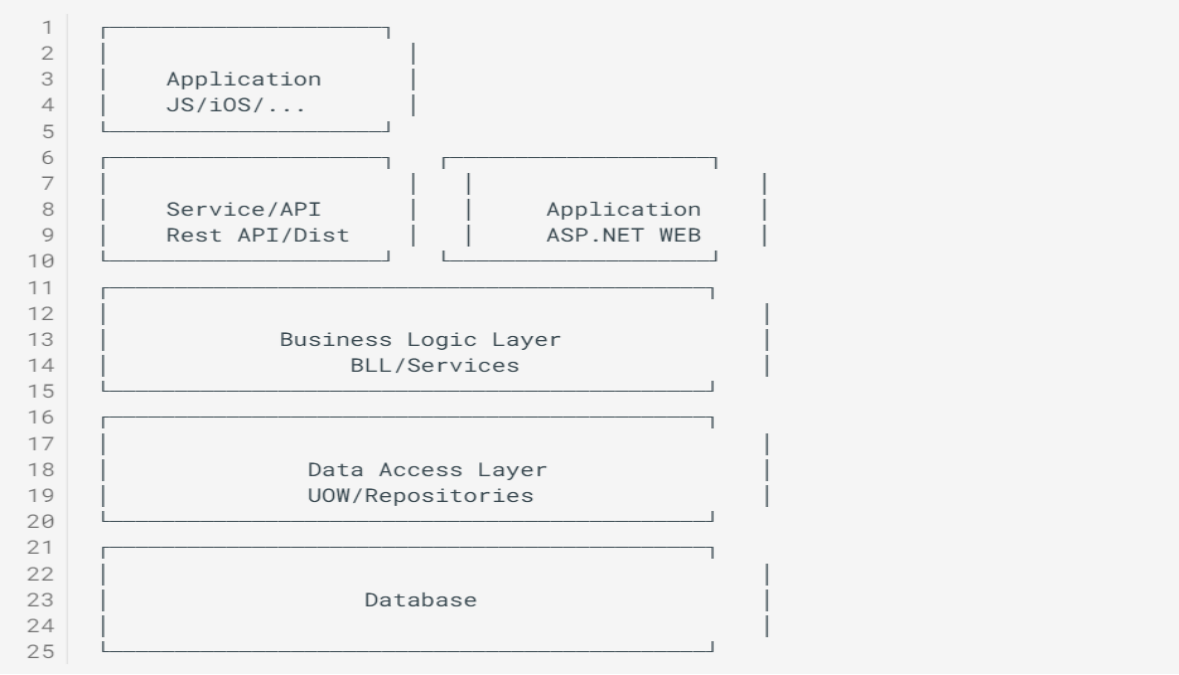
Serveripool kasutati .NET Framework raamistikku. Kliendipoole arendamiseks valiti React, sest see on lihtne ja tõhus raamistik dünaamiliste veebirakenduste loomiseks. Andmebaasina kasutati PostgreSQL andmebaasisüsteemi, sest see on vabavaraline ja pakub suurt hulka toetatud andmetüüpe ja indekseid.

4 Veebirakenduse disain

Veebirakendus tehakse kahes osas: veebiteenus ja klient. Klient hakkab veebiteenuselt päringute kaudu andmeid küsima ning siis neid välja näitama. Mõlemad on arendatud teineteisest sõltumatuna ehk mõlemat saab vajaduse korral välja vahetada nii, et teisega midagi ei juhtu.

4.1 Veebiteenuse-poolne lahendus

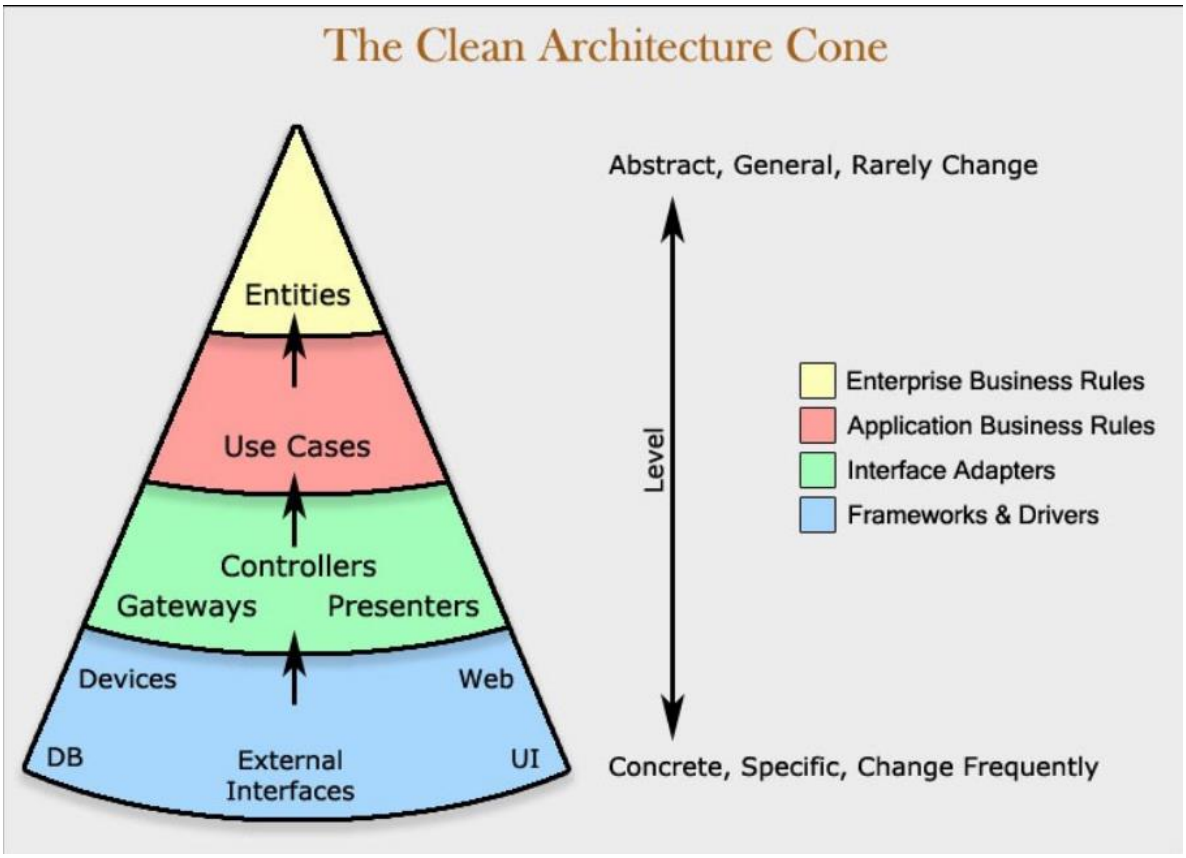
Veebiteenuse-poolne osa lahendatakse hajutatud rakenduse arhitektuuriga, kus paigutatakse eri rakenduse osad üksteisest eraldi ning siis need seotakse omavahel. Rakendus jagatakse alguses kahte lahendusse: baaslahendus ja rakendus ise. Baas sisaldab liideseid ning baas implementatsioone klassidest, mis kas pärivad liidest ning siis rakenduse klassid pärides baas klassi saavad endale nõutud väljad ja funktsioonid juurde või mida tuleb rakenduse lahenduses otse üle kirjutada. Ning rakendusse implementeeritakse andmebaasi osa, andmete juurdepääsu kiht ja äriloogika kiht. Siis saadetakse vajalikud teenused ja funktsioonid API lõpp-punktidele. Joonis 4 näitab veebiteenuse kihte.



Joonis 4. Veebiteenuse kihid [24].

4.1.1 NET rakenduse arhitektuur

Arhitektuurina kasutatakse tarkvara disaini nimega *Clean Architecture*, mis on tarkvara kujundamise filosoofia. See eraldab disaini elemendid ringitasanditeks, mille eesmärgiks on anda arendajatele juhised korraldada koodi nii, et see hõlmaks äriloogikat, kuid hoiaks seda tarnemehhanismist eraldi [25] (Joonis 5).



Joonis 5. *Clean Architecture* disaini koonus [26].

Rakendus on jaotatud erinevatesse projektidesse, mis on omavahel seotud. Projektid on järgmised:

- App.Domain – Domeeniklasse sisaldav projekt
- App.DAL.DTO – Andmetele juurdepääsu kihi DTO (*Data Transfer Object*) ehk andmeedastusobjekte sisaldav projekt
- App.DAL.EF – Projekt, kust saadakse ühendust andmebaasiga, tehakse hoidlad ning kaardistatakse domeeniobjektid andmeedastusobjektideks
- App.BLL.DTO – Äriloogika andmeedastusobjekte sisaldav projekt

- App.BLL – Projekt, kus kirjutatakse äriloogika teenused ja kaardistatakse andmete juurdepääsu kihi andmeedastusobjektid äriloogika andmeedastusobjektideks
- App.Contracts.DAL – Projekt, kus kirjutatakse liidesed andmete juurdepääsu kihile
- App.Contracts.BLL – Projekt, kus kirjutatakse liidesed äriloogikale
- App.Public.DTO – Projekt kus defineeritakse lõppobjektid ning kus toimub objektide versiooneerimine
- App.Public – Projekt, kus kaardistatakse lõppobjekte ja äriloogika objekte
- WebApp – Projekt, kus on defineeritakse API kontrollerid, konfiguratsioonifail, autentimine ja rakenduse sätted

4.1.2 Andmebaas ja selle teostus teenus

Analüüsi käigus selgus, et arendusel mõistlikuks andmebaasi valikuks on PostgreSQL, millel on .NET rakenduses hea toetus. Selleks tuleb rakenduses alla laadida .NET ametlikust paketihaldussüsteemist NuGet PostgreSQL pakett.

Et andmebaas oleks kergelt hallatav on mõistlik kasutada Dockerit, platvorm, mis on loodud et aidata arendajatel kaasaegseid rakendusi luua, jagada ja käivitada. Dockerit kasutades on arendajatel vaja vähem süsteemi seadistada, mis jätab rohkem aega koodi kirjutamiseks. Docker lubab andmebaasi koos kasutajaga ja põhisätetega hõlpsalt püsti panna.

Rakendusse tuleb lisada fail, mis sätib Dockeri üles ning muuta konfiguratsioonifailis andmebaasi teostus teenus PostgreSQL-ks. Viimaks tuleb rakenduse sätetes seada õige ühendussõne.

4.1.3 REST API

REST API on rakenduste liides, mis vastab REST-i arhitektuurilise stiili piirangutele ja võimaldab suhelda RESTful veebiteenustega [27]. Diplomitöö rakenduses defineeritakse kõik API päringud märkides ette /api ning need on jaotatud erinevate ressursside kontrolleritesse. Kontrollerid täpsustavad toimingud sõltuvalt päringu tüübis. Päringu tüübid on järgmised: GET, POST, PUT ja DELETE. Igasse päringusse tuleb lisada ka versiooninumber. Päringuvastus on tagastuskood, diplomitöö rakenduses kasutatakse kõige sagedamini järgmisi tagastuskoode:

- 200 – OK – tähendab, et päring õnnestus
- 201 – Created – tähendab, et uus ressurss on loodud
- 204 – No Content – tähendab, et vastuses pole mõtet saata andmeid, seega saadetakse ainult päis
- 401 – Unauthorized – tähendab, et kasutajal pole õigust antud tegevust sooritada
- 404 – Not Found – tähendab, et serveril ei õnnestunud leida päritud ressurssi
- 500 – Internal Server Error – tähendab, et serveris juhtus viga, millega server ei oska midagi teha

4.1.4 Veebiteenuse turvalisus

Kasutaja autentimiseks kasutatakse standardit JSON Web Token (JWT). Seda kasutatakse veebitööriistade ja rakenduste autentimiseks ning koosneb kolmest osast: päis, kasutajainfo ja allkiri ehk üks JWT näeb tüüpiliselt välja „xxxxx.yyyyy.zzzzz“. Päises hoitakse teavet krüpteerimisest, mida JWT kasutab ning allkirja algoritmi. Kasutajainfo sisaldab autentimiseks vajalikku kasutajainformatsiooni, allkiri genereeritakse JWT loomise ajal ning kinnitab, et kasutaja informatsioon on kehtiv. Allkirja genereerimisel kasutatakse salajast võtit, mida teab ainult server. Joonis 6 näitab koodijuppi JWT genereerimisest, mis antakse kasutajale sisselogimisel kaasa.

```
var jwt = IdentityExtensions.GenerateJwt(
    claimsPrincipal.Claims,
    _configuration["JWT:Key"],
    _configuration["JWT:Issuer"],
    _configuration["JWT:Issuer"],
DateTime.Now.AddMinutes(_configuration.GetValue<int>("JWT:Exp
ireInMinutes"))
);
var res = new JwtResponse()
{
    Token = jwt,
    RefreshToken = refreshToken.Token,
    FirstName = appUser.FirstName,
    LastName = appUser.LastName,
    IsAdmin = appUser.IsAdmin,
};
return Ok(res);
```

Joonis 6. JWT genereerimine ja kasutajale sisselogimisel kaasaandmine.

Antud diplomitöös kasutatakse JWT-d kolmekümne minutilise aegumisajaga ning kui aegumisaeg möödub ja kasutaja pole selle aja jooksul välja loginud, siis uuendatakse JWT ja aegumisaeg hakkab uuesti.

Samuti on mõeldud Brute-force attack-i peale, mis on häkkimise meetod, kui proovitakse järjest esitada võimalikud kasutajanime ja parooli kombinatsioone, et süsteemi sisse saada. Et seda ennetada, on sisselogimisel ja registreerumisel pandud viivitusaeg, mis on suvaliselt valitud 100-1000 millisekundi vahemikus.

4.1.5 Konfiguratsioon

Põhikonfiguratsioonifailiks on veebiteenuses fail nimega Program.cs. Seal konfigureeritakse andmebaasi, süstitakse olemasolevad teenused, lisatakse kohandatud kasutajaklassid jne. Põhikonfiguratsioonis lubatakse ka CORS (*Cross-Origin Resource Sharing*). CORS lubab näidata ressursse, mis ei ole serveri enda omad. Samuti on konfiguratsioonisätted, mis on JSON formaadis, kus on kirjeldatud andmebaasisõne, JWT, andmete initsialiseerimine, logimine jms.

4.1.6 Valideerimine

.NET sisaldab valideerimisraamistikku, mis võimaldab määrata valideerimisreegleid andmetüüpidele ja teostada valideerimist rakenduse koodis. Antud diplomitöös kasutatakse valideerimisereegleid domeeniklassides, kus sätitakse minimaalsed ja maksimaalsed lubatud pikkused sõnedel ning maksimaalsed ja minimaalsed arvud numbritel. Joonis 7 näitab koodijuppi sellest, kuidas seadistatakse valideerimist sõnele, määrates maksimaalse tähtede pikkuse antud väljale.

```
[MaxLength(128)]
public string Name { get; set; } = default!;
```

Joonis 7.Sõne valideerimine C# keeles.

4.2 Klientrakendusepoolne lahendus

Kliendipoolne lahendus on täiesti iseseisev veebiteenusest. Analüüsi käigus sai kliendipoolse lahenduse tarkvaraks valitud React, mis on JavaScripti raamistik. See on väga populaarne

raamistik just oma kiiruse poolest. Reacti eristab teistest see, et eksisteerib ainult ühepoolne andmete sidumine ehk andmeid uuendatakse kohe kui need muutuvad allikas [28].

Reactis on *stateful* ja *stateless* komponendid, mis tähendab et nad kas on olekuga, mida saab mõjutada või nad on olekuvabad. React kasutab märgistuse süntaksit nimega JSX (JavaScript XML). See lubab raamistikus kirjutada HTML-i koodi otse rakenduse koodi sisse.

4.2.1 Komponentide planeerimine

Reactis on komponente mõistlik planeerida varakult, et teada, millised peavad olema olekuga ja millised olekuvabad. Näiteks tehakse rakenduses järjend teenusetüüpidest ning kui mõnele teenusetüübile vajutada, avaneb järgmine järjend, kus on kõik seda tüüpi teenused kirjeldatud. Selle jaoks, et ei peaks järjendi avamiseks lehte uuesti laadima, peab antud komponent omama olekut, antud juhul kas avatud olek või suletud olek. Joonis 8 näitab Reacti koodi, mis lubab ilusalongi veebilehel kasutajal valida töötaja kelle kohta tagasisidet jätta. Selleks peab selle eest hoolitseval komponendil olek olema.

```
const [employee, setEmployee] = useState("");
const handleEmployeeChange = (e: any) => {
    setEmployee(e.target.value);
};
```

Joonis 8. Oleku määramine Reactis.

Kui kasutaja valib mõne töötaja veebilehel, kellele tagasisidet jätta, siis kutsutakse funktsioon handleEmployeeChange välja ning määratakse muutuja employee väärtuseks vastava töötaja nimi, mille kasutaja valis.

4.2.2 React – Single Page Application (SPA)

Single Page Application ehk üheleherakendus on veebileht või veebirakendus, mis kirjutab dünaamiliselt ümber praeguse veebilehe uute andmetega serverist, selle asemel, et veebilehitseja vaikimisi laeb terve lehe uuesti [29].

Üheleherakendus küsib kogu info serverilt korraga ning kui minnakse uuele lehele, siis rakendusel on juba andmed olemas, mida tuleb seal lehel kuvada. Sellega välditakse lehe

uuesti laadimist. Laadimine toimub vaid siis kui andmeid muudetakse. Joonis 9 näitab üheleherakenduse toimimist Reactis.

Joonis 9. Üheleherakenduse toimimine [30].

4.2.3 React rakenduse struktuur

Rakendus on jaotatud erinevatesse kaustadesse, mis on omavahel seotud. Kaustad on järgmised:

- *Components* – rakenduse komponentide ja vaadete kaust, kus on kirjeldatud kõik komponendid, vaated ning on tehtud andmebaasi tabelitele CRUD (Akronüüm: *Create, Read, Update, Delete*) lehed
- *Domain* – domeeniklasside liideseid sisaldav kaust
- *Services* – veebiteenuseid ja nende liideseid sisaldav kaust
- *State* – olekuga komponente kirjeldavat liidest ja andmebaasiga ühendust loova liidest sisaldav kaust

Lisaks sisaldab React raamistikku React Router, mis aitab veebiteenuse ja kliendi navigatsioone juhtida.

4.2.4 Bootstrapi kasutus

Bootstrap on tasuta avatud lähtekoodiga veebiarendusraamistik. See on loodud veebilehtede arendusprotsessi lihtsustama, pakkudes kujunduste süntaksi kogumit [31]. Bootstrapiga saab võtta juba näidisena valmis tehtud kujunduse komponendi ning selle kohe enda veebilehele kopeerida. See aitab veebilehe disainimisel väga palju aega säästa, sest programmeerija peab

palju vähem ise komponente koostama ning CSS (*Cascading Style Sheets*) ja HTML-i koodi kirjutama. Antud diplomitöös kasutatakse Bootstrapi viiendat ehk kõige uuemat versiooni.

4.2.5 React rakenduse serveerimine veebilehitsejale

React kasutab veebilehitsejale rakenduste serveerimiseks Webpacki, mis on mooduli komplekteerija ning seda kasutatakse koos Reactiga sõltuvuste koondamiseks ja haldamiseks. Võetakse kõik üksikud JavaScripti failid ja muud projektis olevad ressursid ning ühendatakse need üheks failiks, mida veebilehitseja saab laadida [32].

4.2.6 Suhtlus veebiteenusega

React suhtleb veebiteenusega läbi Ajax tehnoloogia, mis võimaldab teha asünkroonseid HTTP-päringuid ilma et peaks lehte uuesti laadima. Antud diplomitöös kasutatakse Axios teeki, veebiteenuse API-dega suhtlemiseks. Selleks määratakse URL (*Uniform Resource Locators*), kuhu hakatakse päringuid saatma. Joonis 10 näitab kuidas luuakse HTTP klient, kuhu hakatakse päringuid saatma koos vajalike andmetega ning veebiteenus vastab tagastuskoodidega. Kui päring õnnestus, saadetakse vastavalt kas 200, 201 või 204 tagastuskood, olenevalt sellest, mis päringut tehti. Kood 200 saadetakse kui päring õnnestus ja on vaja kliendile andmeid vastu saata, 201 saadetakse kui päring õnnestus ning oli vaja midagi uut luua ning 204 saadetakse kui päring õnnestus, kuid klient andmeid tagasi ei soovi saada, siis saadetakse ainult päis ilma lisainfota. Kui päring ei õnnestunud, siis saadetakse kas 401, 404 või 500 veakood koos vea kirjeldusega. Kood 401 saadetakse kui kasutajal pole vastavaid õigusi, et antud tegevust sooritada, 404 saadetakse kui ressurss, mida tahetakse näha ei ole leitav ning 500 saadetakse, kui serveril on tekkinud olukord, mida ta ei oska lahendada.

```
let base = "https://localhost:7101/api/v1";
export const httpClient = axios.create({
    baseURL: base,
    headers: {
        "Content-type": "application/json"
    }
})
export default httpClient;
```

Joonis 10. HTTP klient, kuhu klientrakendus hakkab päringuid saatma.

4.2.7 Turvalisus klientrakenduses

Sarnaselt veebiteenusele kasutatakse JWT-d kasutaja autentimisel. Erinevus on selles, et JWT salvestatakse lokaalsesse mällu ning kui kasutaja laeb lehte uuesti, siis otsitakse JWT-d lokaalsest mälust, mitte ei kanta seda päringuga kaasas. Rakenduse turvalisuseks kasutatakse sisemisi teeke ja tehnikaid, mis väldivad andmete lekkimist. Rakenduse HTTP klient on Axios teek, mis loob turvalise HTTPS ühenduse veebiga. Andmeid hoitakse käitusajal komponentides, mis väldib nende lekkimist DOM-i.

4.3 Testimine

Et tagada veebirakenduse korrektne toimimine ja stabiilsus tuleb seda testida. Antud diplomitöös kasutati manuaalset testimist ehk paluti mitmel inimesel ja ilusalongi töötajal erinevaid funktsionaalsusi veebilehel läbi proovida, et näha, kas rakenduse funktsionaalsus toimib ootuspäraselt ning samuti et näha, kas rakendus muutub mõnda operatsiooni tehes aeglasemaks. Testimisel selgus, et rakenduse funktsionaalsused toimivad ja rakendus on stabiilne. Leiti vead administraatori kasutajana andmete lisamisel nagu näiteks liiga kitsad komponendid pika teksti jaoks ehk kui administraator lisab mõnele tootele või teenusele pika kirjelduse, siis see tekst ulatub komponendist välja.

5 Hinnang loodud veebirakendusele

Antud veebirakendus loodi eesmärgiga pakkuda ilusalongi klientidele ja potentsiaalsetele klientidele tõhusat, turvalist ja kasutajasõbralikku lahendust, mis vastab ilusalongi ärivajadustele ja eripäradele. Veebirakenduse serveriosa arendamiseks kasutati C# programmeerimiskeelt, et tagada usaldusväärset andmeedastust ja turvalisust. Kliendipoolne osa on arendatud JavaScripti raamistikus React, mis võimaldab kasutajatel saada sujuvat ja dünaamilist kasutajakogemust. Veebirakenduse arendamisel oli rõhk pandud puhta, kvaliteetse ja hästi organiseeritud koodi kirjutamisele, mis lihtsustab tulevikus luua uusi funktsionaalsusi ning teha rakenduses muudatusi.

Antud veebirakendus annab veebilehe külastajatele mugava ja atraktiivse keskkonna, kus saab tutvuda ilusalongis pakutavate iluteenustega ja kosmeetikatoodetega. Rakendus on loodud skaleeruvust ja paindlikkust arvestades, mis annab võimaluse rakendusele kergesti uusi funktsionaalsusi lisada ja toetab ettevõtte kasvu.

Turvalisuse tagamiseks on rakenduses kasutatud JWT autentimist, mis kaitseb ettevõtte äriandmeid ja klientide isikuandmeid.

Rakendus saavutati nõutud toote tasemele, mis tähendab, et nii kasutajad, kui ka töötajad saaksid rakendust juba kasutada. Kõik ettenähtud funktsionaalsed nõuded said täidetud. Veebilehel saab näha infot teenuste, toodete, allahindluste ja töötajate kohta, saab sisse logida või ennast kasutajaks registreerida, kasutajad saavad jätta hindeid töötajatele, kuid et toode oleks konkurentsivõimeline, peab seda veel paremaks muutma.

5.1 Kasutatud tehnoloogiate uudsus

Analüüsi käigus iga tehnoloogia valiku osas arvestati lisaks teistele aspektidele ka kasutajate hulka ning nende tehnoloogiate uudsust. Kuna tegemist on veebirakendusega, siis kasutati võimalikult populaarseid teeke, lahendusi ja programmeerimiskeeli. Kasutati LTS (*Long Term Support*) versioone programmeerimiskeeltest, mis tähendab et nende toetus jätkub veel pikka aega. Saab järeldada, et kasutatud tehnoloogiad on väga aktuaalsed ja sobivad.

5.2 Võimalused edasi arenduseks

Võimalusi rakenduse edasiarendamiseks on palju, kuid eelkõige oleks eesmärgiks teha rakendus konkurentsivõimeliseks. Selleks peaks välja mõtlema ühele ilusalongile sobiva veebilehe disaini ning lisada keelevahetuse funktsionaalsus. Samuti tuleks läbi viia ka põhjalikum rakenduse testimine, nimelt teha ühik- ja integratsioonitestid, et tagada rakenduse turvalisus, stabiilsus ja usaldusväärsus.

Võimalikud hilisemad edasiarendused on rakendusele broneerimissüsteemi arendamine, rakendusele e-poe arendamine, läbi sotsiaalmeedia registreerimine ja mobiilirakenduse loomine.

6 KOKKUVÕTE

Diplomitöö eesmärgiks oli luua ilusalongile usaldusväärne, kergesti mõistetav ning informatiivne veebirakendus. Lahendus sisaldab endas turvalist kasutajasüsteemi, infot ilusalongi-, töötajate-, pakutavate teenuste ja toodete kohta ja tagasiside jätmise võimalust.

Antud diplomitöö kirjeldab veebirakenduse loomist kahest osast: veebiteenus ja klient. Veebiteenusepoolne lahendus kasutab hajutatud rakenduse arhitektuuri, jaotades rakenduse baaslahendusse ja rakendusse endasse. Arhitektuurina kasutatakse *Clean Architecture* disaini. Rakendus on jagatud erinevateks projektideks, mis tegelevad erinevate ülesannetega, nagu domeeniklassid kirjeldavad andmebaasi tabeleid, andmete juurdepääsu kiht saab ühendust andmebaasiga, äriloogika kirjeldab veebirakenduseks vajaminevat funktsionaalsust. Andmebaasiks valiti PostgreSQL, mis on .NET rakendustes hästi toetatud. Veebiteenuse turvalisus tagatakse JWT autentimisega ning arvestati ka võimalike rünnakutega, mille joaks näiteks suurendati sisse logimisel lehe reaktsiooniaega.

Klientrakendusepoolne lahendus on iseseisev ja kasutab JavaScripti raamistikku React. Rakendus disainiti olema üheleherakendus (SPA), mis võimaldab dünaamiliselt lehte uuendada ilma et peaks lehte uuesti laadima. Rakendus on jaotatud erinevatesse kaustadesse, mis tegelevad erinevate ülesannetega, nagu komponendid, domeeniklassid, veebiteenused. Lisaks kasutatakse React Routerit navigatsiooni juhtimiseks ja Bootstrapi veebilehe disainimisel.

KASUTATUD ALLIKAD

[1] Refactoring Guru. (05.04.2023). Design Patterns. Loetud aadressil https://refactoring.guru/design-patterns

[2] Codecademy Team. (05.04.2023). What is REST? Loetud aadressil https://www.codecademy.com/article/what-is-rest

[3] Codecademy Team. (05.04.2023). Saadaval https://raw.githubusercontent.com/Codecademy/articles/0b631b51723fbb3cc652ef5f009082aa71916e63/images/rest_api.svg

[4] Käver.A. (06.04.2023). Repository Pattern. Loetud aadressil https://courses.taltech.akaver.com/distributed/lectures/08-repository/#repository-pattern

[5] Käver. A. (06.04.2023). Saadaval https://courses.taltech.akaver.com/distributed/images/lecture-08/repo.png

[6] Käver.A. (06.04.2023). Generics – Benefits. Loetud aadressil https://courses.taltech.akaver.com/distributed/lectures/07-generics/#generics-benefits

[7] Upson, M. (07.04.2023). How Many Coding Languages Are There? Loetud aadressil https://www.bestcolleges.com/bootcamps/guides/how-many-coding-languages-are-there/#:~:text=Today%2C%20various%20sources%20report%20anywhere,coding%20sstyl%2C%20and%20intended%20use.

[8] Coursera. (07.04.2023). What Is Python Used For? A Beginner's Guide. Loetud aadressil https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[9] Mozilla. (07.04.2023). What is JavaScript? Loetud aadressil https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[10] Coursera. (07.04.2023). What Is Java Used For? Loetud aadressil https://www.coursera.org/articles/what-is-java-used-for

[11] Microsoft. (07.04.2023). A tour of the C# language. Loetud aadressil https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[12] Oracle. (07.04.2023). What is Ruby? Loetud aadressil https://developer.oracle.com/learn/technical-articles/what-is-ruby#:~:text=Although%20Ruby%20is%20probably%20most,highly%20versatile%20and%20portable%20language.

[13] Petzold. C. (04.12.2006). .Net Book Zero. Saadaval http://www.charlespetzold.com/dotnet/.

[14] Goodman. D. JavaScript Bible 4th Edition, Hungry Minds, 2001

[15] W3Techs. (10.04.2023). Comparison of the usage statistics of React vs. Vue.js vs. Angular for websites. Loetud aadressil https://w3techs.com/technologies/comparison/js-angularjs,js-react,js-vuejs#:~:text=How%20to%20read%20the%20diagram,library%20market%20share%20of%204.1%25.

[16] Smallcombe. M. (10.04.2023). SQL vs NoSQL: 5 Critical Differences. Loetud aadressil https://www.integrate.io/blog/the-sql-vs-nosql-difference/

[17] Talend. (11.04.2023). What is MySql? Everything You Need to Know. Loetud aadressil https://www.talend.com/resources/what-is-mysql/#:~:text=MySQL%20is%20a%20relational%20database,information%20in%20a%20corporate%20network.

[18] Smallcombe M. (11.04.2023). PostgreSQL vs MySQL: The Critical Differences. Loetud aadressil https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/

[19] Tutorialspoint. (13.04.2023). SQLite – Overview. Loetud aadressil https://www.tutorialspoint.com/sqlite/sqlite_overview.htm#:~:text=What%20is%20SQLite%3F,configure%20it%20in%20your%20system.

[20] Hughes. A. (13.04.2023). Microsoft SQL Server. Loetud aadressil https://www.techtarget.com/searchdatamanagement/definition/SQL-Server#:~:text=Microsoft%20SQL%20Server%20is%20a,applications%20in%20corporate%20IT%20environments.

[21] Kanjilal. J. (13.04.2023). Visual Studio vs JetBrains Rider: A Detailed Comparison. Loetud aadressil https://codingsight.com/visual-studio-vs-jetbrains-rider-a-detailed-comparison/

[22] Inouye. J. (13.04.2023). WebStorm vs VS Code: Compare topp IDEs. Loetud aadressil https://www.techrepublic.com/article/webstorm-vs-vscode/#:~:text=WebStorm%20is%20generally%20more%20efficient,unit%20testing%20ensures%20product%20quality.

[23] Howard. (14.04.2023). Virtual Server vs. Physical Server: 8 Key Differences to know. Loetud aadressil https://community.fs.com/blog/Virtual-Server-vs-Physical-Server-8-Key-Differences-to-Know.html

[24] Käver. A. (15.04.2023). Distributed application architecture (N-Tier). Saadaval https://courses.taltech.akaver.com/distributed/lectures/01-http/#distributed-application-architecture-n-tier

[25] Käver. A. (15.04.2023). The Clean Architecture Cone. Loetud aadressil https://courses.taltech.akaver.com/distributed/images/lecture-01/clean-arch.jpeg

[26] Käver.A. (17.04.2023). Saadaval https://courses.taltech.akaver.com/distributed/images/lecture-01/clean-arch.jpeg

[27] Richardson. L. ja Ruby. S. RESTful Web Services, O'Reilly Media, 2007

[28] Imoh.C. (17.04.2023). How To Bind Any Component to Data in React: One-Way Binding. Loetud aadressil https://www.telerik.com/blogs/how-to-bind-any-component-data-react-one-way-binding#:~:text=One%2Dway%20binding%3A%20this%20is,provider)%20automaticalau%20update%20the%20other.

[29] Lawson. K. (18.04.2023). What Are Single Page Applications and Why Do People Like Them So Much? Loetud aadressil https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application

[30] Patel. J. (18.04.2023). Saadaval https://www.monocubed.com/wp-content/uploads/2020/10/How-does-Single-Page-Application-Works.jpg

[31] Jordana. A. (20.04.2023). What Is Bootstrap? Loetud aadressil https://www.hostinger.com/tutorials/what-is-bootstrap/

[32] Apoorva. (20.04.2023). Webpack in React. Loetud aadressil https://www.scaler.com/topics/react/webpack-in-react/

Lisa 1

Mina, Kristian Alex Laherand

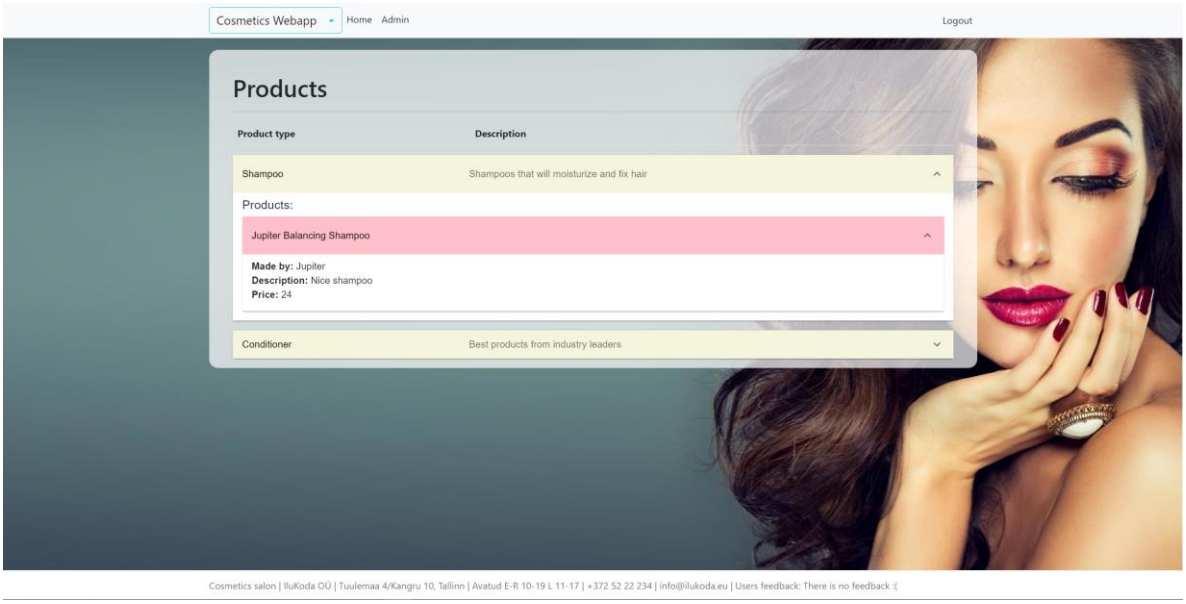
1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose “Ilusalongi veebirakenduse prototüüp”, mille juhendaja on Meelis Antoi
    1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil 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.

Lisa 2 – Teenuse ja klientrakenduse versioonihaldus

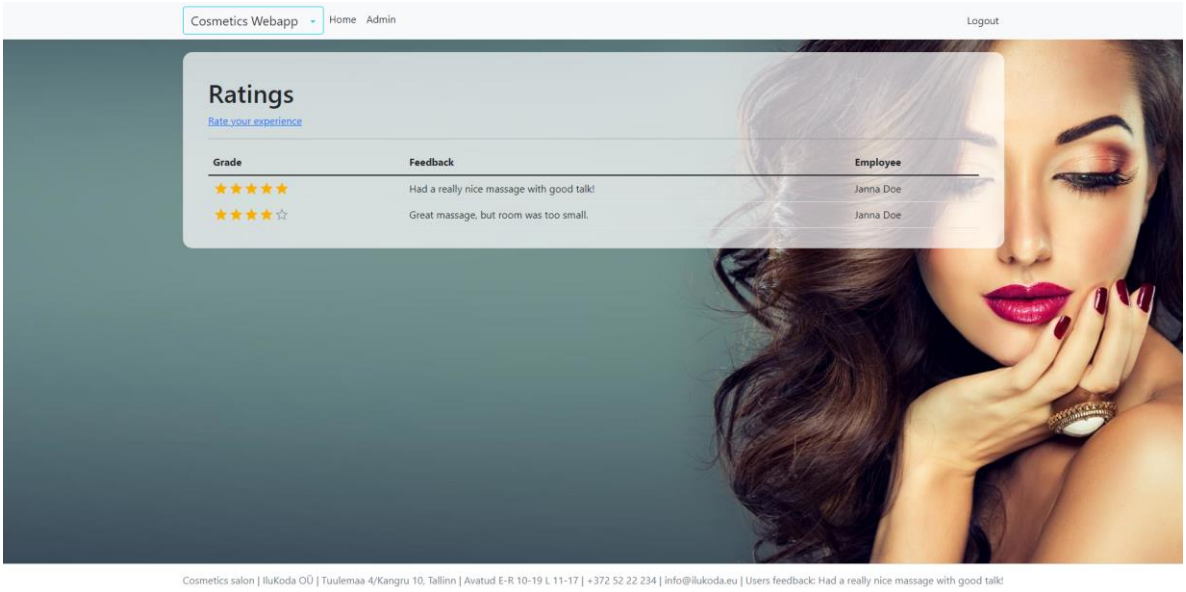
Veebiteenuse kood: https://gitlab.cs.ttu.ee/krlahe/final_thesis_cosmetics_webapp

Klientrakendusepoolne kood:
https://gitlab.cs.ttu.ee/krlahe/final_thesis_cosmetics_webapp_frontend

Lisa 3 – Rakenduse toodete vaade



Lisa 4 – Töötajate tagasiside lehe vaade



Lisa 5 – Rakenduse administraatori paneel