LAB University of Applied Sciences

Digitaalinen hyvinvointimittari

LAB-ammattikorkeakoulu
Liiketalous (AMK), Digitradenomi
2020
Mikko Kaipainen, Timo Leppänen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Kaipainen, Mikko Leppänen, Timo	Opinnäytetyö, AMK	2020
	Sivumäärä 33	

Työn nimi

Digitaalinen hyvinvointimittari

Tutkinto

Tradenomi (AMK)

Ohjaavan opettajan nimi, titteli ja organisaatio

Liisa Uosukainen, lehtori, Liiketalous

Toimeksiantajan nimi, titteli ja organisaatio

-

Tiivistelmä

Opinnäytetyö toteutettiin toiminnallisessa muodossa, jonka osa-alueina rakennettiin full stack-sovellus, johon kuuluu REST-rajapinta, SQL relaatiotietokanta ja Angular web-applikaatio sekä kirjallinen työ.

Työn aiheena oli hyvinvointimittari, pohjautuen aiemmin opinnoissa projektityönä valmistettuun sovellukseen. Hyvinvointimittarin tarkoituksena on tuottaa käyttäjälleen kaavio oman elämänsä eri osa-alueista. Mittari toimii niin, että käyttäjä vastaa kysymyksiin eri aihealueista valitsemalla yhden kolmesta vaihtoehdosta. Vastaamisen jälkeen sovellus tuottaa käyttäjälle tarkasteltavaksi kaavion vastatuista kysymyksistä. Sovellus ottaa myös huomioon vastaamatta jääneet kysymykset ja poistaa ne kaavioilta. Aiempi sovellus, johon tämän opinnäytetyön sovellus pohjaa, tehtiin asiakastyönä hankkeelle ja on julkisen sektorin käytössä. Tätä työtä varten tehty sovellus on paranneltu ja laajennettu toisinto aiemmasta sovelluksesta oppimisen osoittamiseksi.

Varsinainen kirjallinen työ, pitää sisällään teoriaosan ja sovelluksen toimintaperiaatteen selittävän osan sekä johtopäätökset ja jatkokehitys mahdollisuudet. Teoriaosassa perehdytään sovelluksiin ja teknologioihin, joita työssä käytettiin. Sovelluksen toimintaperiaatteet selittävässä osassa kerrotaan perusteellisesti eri full stack-sovelluksen osa-alueista ja pyritään perustelemaan tehdyt valinnat.

Asiasanat

Full stack, SPA, REST, sovellus, rajapinta, Front end, Back end, Angular

Abstract

Author(s)	Type of Publication	Published
Kaipainen, Mikko Leppänen, Timo	Thesis, UAS	2020
	Number of Pages 33	

Title of Publication

Digital wellbeing application

Name of Degree

Bachelor of Business (UAS)

Name, title and organization of the supervising teacher

Liisa Uosukainen, Lecturer, Business

Name, title and organization of the client

-

Abstract

This thesis was done as a functional execution, which contains literary work and Full Stack application including REST API, SQL relational database and Angular web application.

Topic for the application was to produce an application for surveying and tracking wellbeing of individuals with physical and mental deficiencies. The application was based on a similar application made as an earlier school project. The purpose of the application is to produce informational charts from different topics of the user's life. The application works by asking questions and user selecting one of three options. After answering, the application produces charts for the user to review. It considers questions left unanswered and removes them from the chart. Previous application, which this thesis is based on, was made as a project for a client in the public sector. New application expands and improves upon the previous project to indicate learning and individual growth as students.

Written work, which serves as the thesis paper consists of theory and applications functional principles ending with conclusions and further development. Theoretical section focuses on software and technologies used in the functional part of the thesis. The part that seeks to justify functional principles aims to explain different components and the chosen functionalities in the Full stack application.

Keywords

Full stack, SPA, REST, application, interface, Front end, Back end, Angular

Sisällys

1 Johdanto ... 1
1.1 Opinnäytetyön tavoitteet ... 2
1.2 Opinnäytetyön rajaukset ... 3
2 Selvitys Analyysityökaluista ... 4
2.1 Metabase ... 4
2.2 Google Data Studio ... 5
2.3 KNIME Analytics Platform ... 5
2.4 Chartio ... 6
2.5 Microsoft Power BI Embedded ... 7
2.6 Chart.js ... 7
3 Teknologiat ... 9
3.1 SQL ja relaatiotietokanta ... 9
3.2 .NET ja C# ... 9
3.2.1 C# ... 9
3.2.2 .NET ... 10
3.3 Angular ja Node.js ... 11
3.4 Single-Page-Application ... 12
3.5 Node package manager, npm ... 12
3.6 JSON Web Token ... 13
3.7 Sovellukset ... 13
3.7.1 Adobe XD ... 13
3.7.2 Postman ... 14
3.7.3 Visual Studio 2019 ... 14
3.7.4 SQL Server Management Studio ... 15
3.7.5 Webstorm ... 15
3.7.6 Azure ja Azure Devops ... 16
3.7.7 Git ja GitHub ... 17
4 Arkkitehtuurit ... 18
4.1 REST-rajapinta ... 18
5 Valmis sovellus ... 20
5.1 Sovelluksen rakenne ... 20
5.1.1 Web-applikaatio ... 20
5.1.2 REST-rajapinta ... 23
5.1.3 Tietokannat ... 24

5.1.4 Azure ja Azure Devops 25
5.2 Työn ongelmat ja niiden ratkaisut 25
6 Johtopäätökset 27
6.1 Sovelluksen nykytila 27
6.2 Dokumentointi ja työpäiväkirjat 27
6.3 Aikataulutus ja yhteistyö 29
6.4 Erot mittareiden välillä 29
6.5 Sosiaali- ja terveydenhuollon tarve ja nykytila 30
7 Jatkokehitys 31
Lähteet 32

1 Johdanto

Opinnäytetyönä toteutetaan hyvinvointisovellus ja sen raportointi kirjallisena työnä. Hyvinvointisovellus on jatkumo opintojen aikana suoritetulla kurssilla luodulle samankaltaiselle sovellukselle. Aiempi sovellus tehtiin osana Naapurit-hanketta, jota varten oli luotu sovelluksessa käytettävä sisältö valmiiksi. Sovellus, joka hankkeelle tehtiin, ei täyttänyt kaikkia tavoitteita, joita sille projektin alussa asetettiin ja oli toiminnoiltaan vajavainen. Projektiin varattu aika ei myöskään sallinut alkuperäisten vaatimusmäärittelyiden kokonaisvaltaista toteuttamista.

Opinnäytetyöhön ajatus alkuperäisten vaatimusmäärittelyiden mukaisen sovelluksen rakentamisesta oli houkutteleva ja siihen tartuttiin. Siispä opinnäytetyönä tehtävä sovellus on valmistuessaan parempi, laajempi, selkeämpi sekä logiikaltaan järkevämpi kokonaisuus, joka kattaa alkuperäisen suunnitelman sekä uusia matkanvarrella mukaan tulleita ideoita.

Jatko-osa edelliselle sovellukselle ei tule olemaan kaupalliseen käyttöön julkaistava tuote. Sovellus tulee olemaan puhtaasti tekijöiden opinnoissa saadun osaamisen osoittamiseen ja materiaalin keräämiseen tätä paperia varten.

Hyvinvointisovellus on ensisijaisesti suunnattu kehitysvammaisten avustamiseen sekä elämän eri osa-alueiden seurantaan ja niissä kehittymiseen. Lopullinen sovellus ei kuitenkaan rajoita hyvinvoinnin mittaamista ainoastaan kehitysvammaisiin henkilöihin, vaan applikaatio soveltuu mihin tahansa kysymyksin ja vastausvaihtoehdoin mitattavalle aiheelle. Kyseiseen opinnäytetyöhön kohderyhmä valikoitui aiemman sovelluksen kohderyhmän mukaan, koska aiheeseen oli olemassa valmis data testausta varten. Sovelluksen visuaalinen ilme on yksinkertainen ja mahdollisimman neutraali kohderyhmän valinnan ja Angular Materialin tarjoamien mahdollisuuksien mukaisesti.

Sovelluksen toimintaperiaate on yksinkertainen. Sovellukseen kirjaudutaan web-applikaatiossa. Kirjautumisikkuna ohjaa suoraan päävalikkoon, joka näyttää aihealueet. Aihealueita on kymmenen ja ne pitävät sisällään kysymyksiä eri elämän osa-alueista. Yhteen kysymykseen on kohdennettu kolme vastausvaihtoehtoa sekä yksi vapaan sanan kenttä. Vastaukset tallentuvat http-protokollan mukaisesti tietokantaan sekä selaimen sessiovarastoon vastaamisen edetessä. Kysymysten jälkeen käyttäjä ohjataan kaaviona näytettävien vastausten tarkasteluun. Lisäksi sovellukseen on luotu järjestelmänvalvojan työkalut, joilla voi muokata käyttäjiä, kysymyksiä ja aihealueita.

1.1 Opinnäytetyön tavoitteet

Tavoitteena opinnäytetyöllä on osoittaa käytännön osaaminen tehtävän hyvinvointisovelluksen muodossa. Valmiin sovelluksen kautta näkyvät kaikki opitut aiheet opintojen ajalta. Opittuihin aiheisiin kuuluvat full stack-osaaminen, joka pitää sisällään tietokannat, back endin, front endin sekä DevOpsin. DevOps näyttää julkaisun ja jatkuvan toimittamisen toiminnot.

Kirjallisen työn tavoitteena on osoittaa ymmärrys teknologioista, alustoista ja kirjastoista sekä täyden sovelluskokonaisuuden tuottamisen prosessin aina suunnittelusta julkaisuun ja edelleen ylläpitoon.

Kirjallisen työn osa, joka käsittelee valmista sovellusta, käsittää työpäiväkirjojen analysoinnin ja niiden tarkastelun. Tavoitteena on selventää ohjelmointiprosessia ja käydä läpi sovelluksen lopullinen rakenne sekä tuoda ilmi tehdyt ohjelmoinnilliset ratkaisut ja vastata kysymykseen miksi niihin on päädytty. Lisäksi osiossa paneudutaan aikatauluttamiseen ja pohditaan asetettujen aikamääreiden saavuttamisen onnistumista. Osio myös tuo ilmi ongelmat ja niiden ratkaisut.

Kirjallisen työn teoriaosa käsittää sovellukseen käytettävät teknologiat, arkkitehtuurit sekä selvitystyöt. Teoriaosa työssä tulee olemaan laaja full stack sovelluksen luonteen vuoksi. Sovelluksen laajuus itsessään ei ole suuri, mutta sen tekoprosessi sisältää useita eri teknologioita, sovelluksia ja arkkitehtuureja, jotka ovat välttämättömiä avata, jotta tuloksia tarkastelevassa osassa on mahdollista käsitellä kertynyttä aineistoa.

Tavoite tietokannalle on suunnitella ja toteuttaa laaja ja moniulotteinen relaatiotietokanta. Tavoitteena on luoda automaattiset komentorivit tietokannan uudelleenluomista varten.

API-rajapinnan tavoite on tehdä alusta asti suojattu rajapinta noudattaen yleisiä ja hyviä ohjelmointitapoja käyttäen C#-ohjelmointikieltä versiolla 8. Rajapinnan osa-alueet erotellaan selkeästi service-, repository- ja controller-tasoilla, jotta komponentit pysyisivät mahdollisimman kompakteina ja tehokkaina. Lisäksi tavoitteena on tehdä rajapinnasta asynkroninen.

Angular web-applikaatiolle tavoitteeksi asetetaan valmis sovellus, joka on yksinkertainen, niin logiikaltaan kuin visuaaliselta ilmeeltään. Visuaalisen ilmeen tulee olla yksinkertainen, mutta silmää miellyttävä sovelluksen aihe huomioon ottaen. Applikaation toimintalogiikka pyritään pitämään selkeänä. Tämä tavoite saavutetaan, kun komponenttien nimeämiskäytäntö on komponentteja kuvaava, komponentit pidetään kompakteina sekä rakennepuu toteutetaan järkevällä logiikalla. Angular kehystyökalu kehittyy nopeasti ja työn

aloitusajankohtana versiointi oli 9.1.5. Tavoitteena työn edetessä on päivittää sovellus julkaistavaan versioon 10.0.0 ja näin osoittaa myös osaaminen sovelluksen ajankohtaisen ylläpidon osalta.

Sovelluskokonaisuuden dokumentoinnin tavoitteena on dokumentoida sovellus hyvien sovelluskehitystapojen mukaisesti. Dokumentointi on tärkeää jatkokehityksen ja tietoturvan kannalta. Dokumentoinnilla helpotetaan valmiin sovelluksen virheenhallintaa ja päivittämistä.

1.2 Opinnäytetyön rajaukset

Analyysityökalun valinta rajataan kuuteen sopivimpaan työkaluun. Ratkaisu työkalun vallinnasta tehdään hinnan, käytettävyyden ja tietokantadatan analysoinnin yhteensopivuuden suhteen.

Opinnäytetyö rajataan toiminnallisen osan perustoimintoihin. Tietokanta luodaan alustavan mallin mukaan, joka tulee kehitysvammaisten elämäntilanteen hallintaan ja seurantaan, mutta soveltuu tarvittaessa kyselydataa muuttamalla muuhunkin vastaavaan mittaamiseen.

Angular web-applikaation järjestelmänvalvojan työkalut rajataan mahdollisuuteen luoda uusia käyttäjiä, kohdentaa käyttöoikeuksia sekä kysymyksiä ja vastausvaihtoehtoja oikeisiin aihealueisiin.

Analyysityökalujen käyttö rajataan niin, että voidaan suorittaa perushakutoiminnot, kuten sukupuoli, ikä ja vastausten keskiarvot. Lisäksi mahdollisuus verrata eri päivämäärien mukaan kyselyn vastauksia.

API-rajapinta rajautuu luonnollisesti tietokannan laajuuden mukaan, sen toimintojen ollessa lähes vakiot.

2 Selvitys Analyysityökaluista

Sovellusta varten tehtiin analyysityökalujen selvitys. Selvitys tehtiin, koska hyvinvointisovellukseen haluttiin liittää työkalut vastausten tarkkailuun niin yhden istunnon aikana tehtävän kyselyn vastausten kuin pitkän aikavälin vastausten vertailuun.

Analyysityökaluja tarkasteltiin hinnan, käytettävyyden ja tietokantadatan liittämismahdollisuuksien mukaan. Huomioon otettiin myös työkalun hallittavuus sekä sen liittämisen helppous. Lopullisen tarkastelun kohteiksi valikoitui kuusi eri työkalua, jotka ovat samankaltaisia, mutta merkittäviä erojakin löytyi. Merkittävimmät erot liittyivät lähinnä ohjelmoinnillisiin seikkoihin, esimerkiksi työkalun esiasennus ja ennakkovaatimukset, integrointi web-applikaatioon ja tietokannan yhteensopivuus sekä työkalun ominaisuuksien muokkaus.

Sovellusta varten valittiin lopulta Chart.js niminen JavaScript kirjasto, joka on erikoistunut kaavioiden muodostamiseen ja niiden visuaalisen ilmeen muokkaamiseen JavaScriptiä käyttäen. Angular sovelluksen ohjelmointikieli on TypeScript, joka on supersetti JavaScriptille. Näin ollen yhteensopivuus on taattu ja se toimii tarkoitukseen paremmin kuin hyvin.

2.1 Metabase

Ilmainen Javalla tehty ohjelma, joka hakee dataa suoraan tietokannasta. Tukee useaa erilaista tietokantasovellusta, mukaan lukien suurimpia pilvipalveluita. Suurin osa on SQL tyyppisiä, mutta seassa on myös NoSQL tietokantoja. Asennuksessa tulee mukana esimerkki dataa, joka tulee H2 tietokantatyypin muodossa.

Ohjelma käynnistetään komentokehotteessa, eikä se vaadi järjestelmänvalvojan oikeuksia toimiakseen. Ainoa erillinen vaatimus on Java JDK 8+ versio, jonka päälle ohjelma käynnistyy. Normaalikäytäntö Java ohjelmille. Itse paketti syö levytilaa 190 megatavua ja ensimmäisen käynnistyksen jälkeen, kun pääkäyttäjä on luotu eikä mitään muuta, on levyntarve noussut 290 megatavuun.

Ohjelma luo saatavilla olevan datan perusteella valmiita kyselyitä, jotka se näyttää graafeina, taulukoina ja numeroina. Kaikkea voi muuttaa lennossa oman tarpeensa mukaan ja uusien luominen tyhjästä on myös helppoa ja nopeaa. Edes henkilölle, joka ei usein tällaisia tee, tuntui menut ja navigointi hyvin luonnollisilta ja intuitiivisilta.

Ympäristöön saa luotua käyttäjiä erilaisilla käyttöoikeuksilla. Sähköpostin lähetys myös onnistuu suoraan ohjelmasta. Slack integraatio löytyy myös, jonka lisäksi löytyy MetaBot, joka hakee ja tallentaa kysymyksiä suoraan Slackista. Käyttäjien autentikointi tapahtuu

normaalilla tunnus ja salasana yhdistelmällä. Lisäksi voidaan käyttää Googlen kirjautumista, jos käyttäjän sähköpostiosoitteet täsmäävät keskenään. Lisäksi voidaan määritellä täysin oma ulkopuolinen autentikointipalvelin, jos esimerkiksi yrityksellä on jo sellainen olemassa entisestään.

Metabase toimii siis erillisenä web-sovelluksena, jota voi käyttää mistäpäin maailmaa tahansa. Asetuksista löytyy erillinen kohta, jossa voi sisällyttää Metabase analytiikan muihin sovelluksiin. Tämä vaatii ilmaiskäytössä tekstin ”Powered by Metabase” sekä Metabase logon sivulle. Näistä pääsee eroon ostamalla erillisen lisenssin. Lisenssi maksaa 300 dollaria kuukaudessa tai 3,000 dollaria vuodessa, jolloin kätevästi säästää 20 % kuluissa. Sen jälkeen voi luoda loputtomasti mitä ohjelmalla vain pystyy tekemään ilman murheita oikeuksista ja Metabasen maininnoista. Isolle yritykselle summa ei ole suuri, mutta tässä opinnäytetyössä aivan liian kallis. Maininnat muutenkaan eivät todennäköisesti haittaa mitään. Lisenssin voi kuitenkin ostaa milloin tahansa ja silloin automaattisesti merkinnät poistuvat, kun lisenssi on maksettu ja koodi vastaanotettu. (Metabase 2020.)

2.2 Google Data Studio

Yhdistää Googlen muut työkalut (Analytics, Ads, BigQuery) kompaktisti samaan pakettiin ja lisää mahdollisuuden tehdä omia yhteyksiä muihin palveluihin. Merkittävä miinus tällaisessa on sen pilvessä toimiminen. Paikalliset tietokannat täytyy avata internetiin päin, jotta Data Studio pääsee käsiksi tietoihin, joka aiheuttaa omat ongelmansa tietoturvan ja henkilötietojen käsittelyn kanssa. Tiedot saa myös kulkemaan API-rajapinnan läpi, jolla pystyy huomattavasti rajoittamaan pääsyä dataan, mutta rajapinta toimii kuitenkin internetissä, joka tässä tapauksessa aiheuttaa pullonkaulan tietoturvassa.

Google Data Studio on ilmainen kaikille käyttäjille, kuten melkein kaikki muutkin heidän tekemänsä sovellukset. Yritysedustajana toimivan täytyy täyttää yhteystiedot kontakteille, joka johtuu Euroopan Unionin tai toisessa tapauksessa Kalifornian lainsäädännöstä. Googlen valvonnan rajallisuuden takia näistä voi pysyä erossa omalla ilmoituksella, ettei ole velvollinen kyseisiin toimenpiteisiin. (Google 2020.)

2.3 KNIME Analytics Platform

KNIME on erittäin tehokas analyysityökalu, jolla voi tehdä melkein mitä vain, miten haluat. Valmiit pohjat toimivat enimmäkseen tutoriaaleina, kuinka ohjelmaa käytetään ja mitä on mahdollista saada aikaan. Kaikki muodostuu pienistä palasista, joita yhdistelemällä saadaan haluttu lopputulos. Tapa kuulostaa hyvältä ja yksinkertaiselta, mikä tässä tapauksessa on ohjelman paras ja samalla huonoin puoli. On äärimmäisen tehokasta voida

itse säätää ja muotoilla kaikki yksityiskohdat, mutta samalla se tekee luomisesta erittäin hitaan ja hankalan. Virheitä sattuu helposti joka vaiheessa, jolloin työaikaa menee hukkaan helposti suuret määrät. Jos olet ammattilainen tämän kanssa, lopputuloksena on vahvoja ja selkeitä analyyseja, mutta ennen sitä minkäänlaiseen analyysiin on vaikea päästä.

KNIME tukee useita eri tietolähteitä, tässä kontekstissa tärkeimpänä mahdollisuus toimia paikallisesti. Valmiista palikoista löytyy tuki yleisesti käytetyille tietokannoille, mukaan lukien mahdollisuus yhdistää Microsoftin Access tietokantaan. Jos käytössä on muu tietokanta, paikallinen tai pilvessä, löytyy vaihtoehto luoda oma yhteys, mutta tällöin tarvitaan tietotaitoa palvelimista ja tietoyhteyksistä.

Rahastusmalli ohjelmalla on erikoinen. Asennuspaketin saa ladata internetsivuilta täyttämällä lomakkeen, jolla kysytään perustietoja kuten nimi, sähköposti, yritys ja kysymys, että mistä kuulit KNIME:n olemassaolosta. Sen jälkeen lataus käynnistyy ja olet ohjelman laillinen käyttäjä. Erikoisen tästä tekee se, että käytössä on myös commercial-lisenssi, jonka voi halutessaan maksaa. Kyse on enemmän yrityksen tukemisesta ja päivitysten sekä parannuksien varmistamisesta tulevaisuudessa. Lisenssin ostoa ei löydy internetsivuilta ollenkaan, vaan ilmeisesti täytyy ottaa heidän myyntihenkilöstöönsä yhteyttä suoraan.

KNIME on myös täysin avoimen lähdekoodin sovellus. Yrityksen mielestä avoimuus ja läpinäkyvyys on paras tapa luoda luottamusta ja uskoa tänä päivänä. Näin jokainen käyttäjä voi tutustua itse, mitä ohjelma tekee ja näin voidaan todistaa, ettei sovellus tee mitään ylimääräistä, kuten tietojen kalastelua tai tietojen lähetystä millekään ulkopuoliselle taholle. (KNIME AG 2020.)

2.4 Chartio

Chartiossa on mahdollisuus tietokantataulujen datasta muodostaa kaavioita ja analyyseja drag ’n’ drop-menetelmällä. Käyttäminen on näin ollen tehty erittäin helpoksi, johon ei välttämättä tarvita yhtään taitoa koodata. Haastavin osuus Chartion käytössä on ennakkoasetusten muokkaaminen. SQL Management Studiossa ja Windowsin omalle palomuurille on tehtävä kokonaan uudet asetukset. Ohjelma toimii internetin yli niin, että Chartion palvelimille on annettava oikeus tulla palomuurin läpi omalle koneelle ja yhdistää tietokantaan. Annettujen oikeuksien jälkeen yhteys muodostetaan Chartion internetsivuilla, josta löytyy myös käyttöliittymä ja työpöytien koodirivit HTML upotuksiin.

Chartion rahastusmalli on kuukausihinnoiteltu, mutta veloitetaan kerran vuodessa. Per asiakkuus täytyy lunastaa vähintään viisi lisenssiä. Malli esitetään kuukausihintaisena vain 40 $/kk, mutta halvimmillaankin palvelu maksaisi 2400 $/vuosi. Vallitsevaan hintatasoon nähden palvelu on kuitenkin edullinen.

Chartiossa on myös mahdollisuus tulostaa analyysit PDF ja CSV tiedostoina, sekä mahdollisuuden kustomoida omaa upotuskoodia. Tietoturvaa luvataan useilla eri leimoilla, joita ovat mm. GDPR, HIPAA, SOC 2 ja Privacy Shield.

Chartio tarjoaa kattavan dokumentoinnin ja oppaat työpöytien hallintaan ja muodostamiseen. (Chartio 2020.)

2.5 Microsoft Power BI Embedded

Microsoft tarjoaa useita eri malleja analysoida dataa. Tähän selvitykseen valikoitui Power BI Embedded, joka soveltuu Microsoftin palveluista parhaiten opinnäytetyön tarpeeseen.

Power BI Embedded vaatii tietotaitoa JavaScript SDK paketeista sekä REST-rajapinnoista. Ensimmäinen vaihe on Microsoftin Storesta ladattavalla työpöytäsovelluksella muodostaa yhteys datalähteisiin, mallintaa data ja luoda interaktiiviset visualisoinnit. Opinnäytetyössä tämä vastaa tarpeeseen, koska kyseessä on Microsoftin työkalu, joka integroituu hyvin Microsoftin SQL Management Studion kanssa ja yhteys tietokantaan on taattu. Toinen vaihe käsittää raporttien julkaisun ja testaamisen JavaScript SDK:lla ja REST-rajapinnalla. Tätä varten pitää rakentaa työympäristö. Lopulta kolmas vaihe vaatii liittämistä varten upotuskoodit. Palveluun autentikoidaan tunnuksen avulla käyttäjät.

Rahastusmalli on joko kuukausittainen perusmaksu 9,99 $ Pro-versiosta tai vuosittainen 4995 $ maksu Premium-versiosta. Molemmissa käytön mukaan tulee lisää hintaa datan määrän mukaan. Lisää kapasiteettia voi hankkia kätevästi Microsoftin pilvipalvelu Azuresta. (Microsoft 2020.)

2.6 Chart.js

JavaScriptillä tehty kirjasto, jolla voi toteuttaa kaavioita reaaliajassa. Yksinkertaisimmillaan pari riviä koodia lisätty sivulle ja ensimmäinen kaavio on valmis. Muokattavissa monipuolisesti erityisesti piirrettävän tyypin ja värien osalta. Tukee kahdeksaa eri kaavio tyyppiä ja värejä voi käyttää millä tahansa yleisesti käytössä olevalla tavalla, kuten RGB-, ja HEX-koodit tai värien nimet. Lisäksi kaaviosta voi värittää eri elementit omilla väreillään ja näin luoda oman näköisen kaavion joka kerta. Tyypeistä löytyy yleisesti tunnettuja ja nähtyjä vaihtoehtoja, kuten viivat, palkit, ympyrät ja muita vähemmän käytettyjä, joille on omat käyttökohteensa.

Dataa kaavioihin voi syöttää kiinteillä arvoilla suoraan lähdekoodiin, hakea palvelinkutsuilla toisilta palvelimilta tai palveluista, sekä käyttäjän syöttämällä datalla. Sisäänrakennetuilla funktioilla voidaan kaavio joko päivittää uudella datalla tai ulkoasulla tai tarpeen vaatiessa,

tuhota nykyinen ja piirtää uusi. Tämä kaikki tietysti tapahtuu millisekunneissa eikä ihmisen silmä sitä edes huomaa. Lisäksi voidaan ottaa käyttöön animaatiot, joka piirtää siirtymävaiheen tyylikkäästi ja sulavasti, jolloin se myös piilottaa mahdolliset uudelleen piirtämisesti johtuvat häiriöt.

Chart.js on avoimen lähdekoodin kirjasto, joka käyttää MIT lisenssiä. MIT lisenssillä kirjasto on ilmainen käyttää, muokata ja julkaista edelleen samalla lisenssillä. Lisensointi pitää olla selkeästi näkyvillä. Näin ollen hinta on erittäin sopiva opinnäytetyö mielessä pitäen. (Chart.js 2020.)

3 Teknologiat

3.1 SQL ja relaatiotietokanta

SQL on standardoitu 1987 kansainvälisellä ISO-standardilla, jota päivitetään viiden vuoden välein. Vakiintuneena kielenä se on saanut pysyvän jalansijan sovelluskehitysmarkkinoilla. SQL on kieli, jolla tallennetaan, haetaan ja manipuloidaan tietoa relaatiotietokannoissa. Se on yleisesti käytetty eikä sen perusosaaminen vaadi paljoa opettelua. Perushaut tietokannoista ovat yksinkertaisia ja riippuen käytetyistä teknologioista ja sovelluksista, sitä ei tarvitse välttämättä osata edes ollenkaan. Kirjastot ja ohjelmistokehykset monesti sisältävät omia, vielä yksinkertaisempia tapoja hoitaa haut ja tallennukset tietokantoihin. Sitä monimutkaisemmat kyselyt vaativat jo osaamista ja tietokantojen ja SQL-kielen tuntemista paremmin. Mahdollisuuksia on todella paljon ja kieli taipuukin monenlaiseen vaativaan käyttöön taitojen ja kykyjen kasvaessa.

Kehitys alkoi 1970-luvulla, joten ominaisuuksia on ehtinyt kerääntyä pitkään ja vastaan tulleet ongelmat on jo ehditty ratkaista. Siksi se onkin yksi yleisimmistä tietokantoihin liitetyistä termeistä.

Relaatiotietokannat, joita SQL:llä muokataan, koostuvat kokoelmista, jotka sisältävät tauluja ja niiden välisiä riippuvuuksia. Taulut sisältävät tietueita, joilla on spesifioidut datatyypit eli attribuutit. Attribuutit määrittävät tallennettavan datan muodon, joita voivat olla esimerkiksi merkkijono (string), kokonaisluku (integer), boolean tai päivämäärä (date). (IBM)

3.2 .NET ja C#

.NET, myöhemmin dotnet, ja C# ovat Microsoftin kehittämät avoimen lähdekoodin kehitysalusta ja ohjelmointikieli. Molemmat ovat hyvin yleisesti käytettyjä ja monipuolisia kehitystyökaluja.

3.2.1 C#

C# pohjaa avoimeen lähdekoodiin ja on Microsoftin 2001 julkaisema ohjelmointikieli. C# on vahvan tyypityksen omaava olio-ohjelmointikieli, jota kuvataan yksinkertaiseksi, moderniksi, joustavaksi ja monipuoliseksi kieleksi. Microsoftin ajatuksena oli kehittää ohjelmointikieli, joka tarjoaa joustavuutta ja ominaisuuksia modernille ohjelmoijalle rakentaa sovelluksia, joiden toiminta on varmistettu pitkälle tulevaisuuteen. Tämän takaa ajatus ohjelmointikielestä, joka on suunniteltu liiketoiminnan ja yritysten tarpeet edellä. Toiminnallisuudet, joita C# tarjoaa, tukevat nykyaikaista ohjelmistokehitystä ja toimii niin web-applikaatioiden, mobiiliapplikaatioiden kuin rajapintojen rakentamisessa.

C# on cross-platform ohjelmointikieli, joten sillä tehdyt .NET sovellukset voidaan julkaista sekä Windows ja Linux että Mac alustoille. C# applikaatiot voidaan myös sijoittaa pilveen tai kontteihin esimerkiksi Dockerilla. C# on kehitetty ohjaamaan ohjelmoijaa kirjoittamaan siistiä, tehokasta, ja turvallista koodia. (Chand, 2020.)

Olio-ohjelmointikielenä C# tukee kapseloinnin, perimisen ja polymorfismin käsitteitä, jolloin luokka voi periä yhden isäntäluokan, mutta voi implementoida useaa rajapintaa. Metodit, jotka yliajavat virtuaalimetodit isäntäluokassa, vaativat avaimen, jolla estetään metodien uudelleenmäärittäminen. C# tekee komponenttien koodaamisen helpoksi usean eri kielirakenteen ansioista, jotka ovat:

- Kapseloidut metodinormit (Encapsulated method signatures), jotka mahdollistavat tyyppiturvalliset tapahtumailmoitukset
- Ominaisuudet (Properties), jotka toimivat yksityisten muuttujien hakuina.
- Attribuutit (Attributes), jotka tarjoavat metadatan tyypeistä koodin ajossa
- XML dokumentaatiokommentointi
- LINQ (Language-Integrated Query), joka tarjoaa sisäänrakennetun kyselymahdollisuuden eri datalähteistä
- Rakennesovittamisen (Pattern matching), joka mahdollistaa sujuvan työnkulun tarkastamalla tietotyypit ja niiden arvot. (Microsoft.)

3.2.2 .NET

Dotnet on avoimen lähdekoodin alustariippumaton-kehitysalusta erityyppisten sovellusten kehittämiseen. Se tukee kolmea eri ohjelmointikieltä, jotka ovat C#, F# ja Visual Basic ja muodostuu kolmesta eri toteutuksesta. Alustariippumattomuus mahdollistaa sillä tehtyjen sovellusten ajamisen millä tahansa yhteensopivalla alustalla. .NET Core on websivujen, serverien, konsoliapplikaatioiden Dotnet-toteutukseen tarkoitettu alusta ja ajaa applikaatioita Windowsilla, Linuxilla ja macOS:llä. .NET Framework tukee esimerkiksi websivuja, palveluja, työpöytäsovelluksia Windowsilla. Xamarin ja Mono ovat dotnet-toteutuksia sovellusten ajamiseen kaikissa suurimmissa mobiilikäyttöjärjestelmissä.

.NET Standard on peruspaketti rajapintoja, jotka ovat samat kaikille dotnet-toteutuksille. Jokainen toteutus sisältää lisäksi rajapintoja liittyen käyttöjärjestelmiin, jolle se on tarkoitettu. Dotnet sisältää myös NuGet paketinhallinnan, joka on tehty erityisesti dotnettiä varten ja sisältää yli 90 000 pakettia. (Microsoft.)

3.3 Angular ja Node.js

Angular on ohjelmistokehys yksisivuisten applikaatioiden kehittämiseen HTML ja TypeScriptin avulla. Se on avoimen lähdekoodiin perustuva MIT-lisensoitu ohjelmistokehys, jonka kehityksestä vastaa useasta henkilöstä muodostuva tiimi.

Angularin arkkitehtuuri nojaa moduuleihin ja komponentteihin. Moduulit ovat rakennuspalikoita, jotka tarjoavat kontekstin komponenttien kokoamiseen. Tavanomainen Angular applikaatio muodostuu useista moduuleista, joista yksi on niin sanottu juurimoduuli, josta applikaatio käynnistyy ja huolehtii itsenäisesti ominaisuusmoduulien käynnistämisen. Ominaisuusmoduuleiksi kutsutaan muita juurimoduulin tukena olevia moduuleja. Moduulit mahdollistavat applikaation lataamisen moduuli kerrallaan, kun niitä tarvitaan. Tällä poistetaan suuren koodimassan lataaminen applikaatiota käynnistäessä.

Angularin komponentit muodostuvat html-, css- ja ts-tiedostoista. Komponentit määrittelevät selaimessa näkyvän visuaalisen ilmeen ja niiden takana olevan logiikan. Angular applikaatiossa on vähintään yksi komponentti, joka yhdistää komponenttihierarkian sivun dokumenttioliomalliin. Komponentti määrittelee luokan, joka sisältää applikaation datan ja logiikan ja on yhteydessä HTML-sapluunaan. Sapluuna taas määrittää millainen näkymä esitetään selaimessa. Data tai logiikka, jota käytetään applikaatiossa yleisesti, injektoidaan komponentteihin palveluista riippuvuuksina, joiden ansiosta samaa dataa voidaan yhdellä palvelulla kohdentaa usealle eri komponentille.

Merkittävässä asemassa Angularissa on reititys, joka määritellään omaan moduuliinsa. Tässä moduulissa luodaan applikaation rakenteelle reitit sekä luvat käyttäjätyypeille. Reititin mahdollistaa navigoinnin komponenttien välillä näkymästä toiseen. Se toimii perinteisellä selaimen navigointimallilla, jossa navigointi tapahtuu joko URL:iin lisäämällä sivun linkki, sivuston omien painikkeiden tai selaimen eteen ja taakse painikkeiden avulla. Angularissa reitin moduuli vastaa tästä kaikesta. Perinteisesti navigointi ja painikkeet toimivat selainhistorian sivuihin pohjautuen, mutta Angular mahdollistaa sen, ettei URL:n muutos lataa aina sivuja uudestaan, vaan moduulit yhden kerran tarpeen tullen. Tästä tulee nimitys yksisivuinen applikaatio.

Angular CLI on komentorivikäyttöliittymätyökalu, jolla alustetaan, kehitetään ja ylläpidetään Angular applikaatioita suoraan komentokehotteesta. Angular CLI tuo mukanaan webpack nimisen niputtamistyökalun, joka hallinnoi kolmansien osapuolien kirjastojen lisäämisen projekteihin. Webpack hallinnoi kirjastojen tuontiprosessin Angular projektiin ja kääntää ne moduuleiksi automaattisesti. Lisäksi webpack sisältää myös työkalun, joka projektiin kirjoitettujen muutosten jälkeen haistelee muutokset ja automaattisesti päivittää selaimen

sivun, joka kuuntelee applikaatiota paikallisesti. Angular CLI asennetaan käyttämällä Node package manageria. (Google.)

Noje.js on asynkroninen palvelin puolen JavaScript-alusta, joka toimii taustalla ja ajaa sovelluksen tapahtumia silmukassa itsenäisesti.

3.4 Single-Page-Application

Single-Page-Application eli SPA tarkoittaa sovellusta, joka toimii nimen mukaisesti yhdellä sivulla tai yhdessä näkymässä. Multi-Page-Applicationeihin eli monella sivulla toimiviin vanhemman ajan tyylisiin sovelluksiin verrattuna käyttäjän kokemus on yleensä yksinkertaisempi ja suoraviivaisempi. Kun dataa näytetään vain vähän kerrallaan ja sitä on saman tien saatavilla lisää, helpottuu selaaminen ja käyttö.

Idea on, että varsinainen sivu rakennetaan ja näytetään käyttäjälle kerran ja vain sisältöä muutetaan tarpeen mukaan. Näin saadaan sulavampi kokonaisuus, jonka toiminnallisuus usein säilyy, vaikka internet yhteys yllättäen katoasi. Tämä johtuu siitä, että data monesti ladataan taustalle odottamaan, eikä kutsuja taustasovelluksiin ja tietokantoihin tehdä jatkuvasti, tilanteen niin vaatiessa. Verkkokaupat ovat hyvä esimerkki, kuinka tehdään yksi toiminto kerralla ja sen jälkeen vasta yritetään edes siirtyä seuraavaan vaiheeseen. Tässä vaiheessa tehdään yleensä hakuja riippuen mitä edellisellä sivulla on syötetty käyttäjän toimesta, kuten postinumero, jotta voidaan tarjota lähellä sijaitsevia postin toimipisteitä tai pakettiautomaatteja.

3.5 Node package manager, npm

Node package manager tai npm, on maailman suurin sovellusrekisteri ja sen omistaa GitHub. Se on tarkoitettu sovelluskehitykseen pakettien lainaamiseen ja jakamiseen tarkoitettu rekisteri. Npm muodostuu kolmesta komponentista, joita ovat websivu, komentorivirajapinta (CLI) ja rekisteri. (npm Inc.)

Websivu tarjoaa mahdollisuudet esimerkiksi pakettien hakuun, profiilien luomiseen. Komentorivirajapinta on käytössä terminaalin kautta ja on yleensä se, jonka sovelluskehittäjä näkee. Rekisteri on laaja julkinen tietokanta JavaScript-ohjelmistoja ja niiden ympärille koottua metadataa. (npm Inc.)

Npm:ia käytetään pakettien liittämiseen sovelluksiin, yksittäisten työkalujen lataamiseen, pakettien ajamiseen lataamatta käyttämällä npx:ää, koodin jakamiseen, koodin rajaamiseen tietyille käyttäjille ja pakettien ylläpidon, koodaamisen ja kehittäjien koordinoimiseen sekä virtuaalisten tiimien muodostamiseen, riippuvuuksien ja

versioidenhallintaan, applikaatioiden reaaliaikaiseen päivittämiseen sekä vertaistukeen. (npm Inc.)

3.6 JSON Web Token

Avoin standardi, jolla on määritelty mahdollisimman kompakti tapa turvallisesti siirtää tietoa käyttäjältä toiselle JSON muodossa. JWT luo HMAC algoritmia käyttäen digitaalisen allekirjoituksen, jonka tarkistus muuttuu, jos sitä yrittää muokata tai väärinkäyttää. Tämä luo turvallisen tavan hoitaa kirjautuminen järjestelmiin. JWT tukee myös tapoja salata avain, jos niin tarvitaan, mutta monessa tapauksessa pelkkä digitaalinen allekirjoitus on riittävä varmistamaan käyttäjän identiteetti.

3.7 Sovellukset

Työhön käytettävät sovellukset valikoituivat opintojen aikana käytettyjen sovellusten mukaan.

3.7.1 Adobe XD

Adobe XD on Adoben 2017 beta-testausvaiheesta poistunut ja tuotantoon siirtynyt suunnitteluohjelma. Adobe XD on käyttökokemus- ja käyttöliittymä suunnitteluun tarkoitettu sovellus, jolla luodaan visuaalisesti näyttäviä prototyyppejä. Prototyypit sisältävät perustoiminnot, mutta eivät tallenna dataa väliaikaisesti eikä pysyvästi. Näin ollen Adobe XD on puhtaasti suunnitteluun tarkoitettu ohjelma.

Ohjelmalla voidaan luoda täydellisiä prototyyppejä käyttöliittymistä riippumatta käyttöjärjestelmästä. Adobe XD taipuu niin web-, työpöytä- ja mobiilisovellusten käyttöliittymien suunnitteluun.

Avainasemassa ominaisuuksissa on niin sanotut taidetaulut, joihin prototyyppi rakennetaan. Tauluja voi muokata web- tai mobiilinäkökulmista. Lisäksi avainominaisuuksia ovat ristikkomallinen taulujen kopiointityökalu, tarvikepaneeli, josta näkee tyylit, värit ja komponentit sekä ääniohjattu objektien muokkaus. Adobe XD:n ominaisuuksiin kuuluvat myös symbolien ja nappien luominen, näppäimistön pikakomennot esimerkiksi objektien koon muuttamiseen, CSS-koodipalikoiden luominen prototyypistä, automaattisesti reagoiva objektien muokkaus taidetauluissa ja suuri määrä integraatiomahdollisuuksia eri alustoilla toimiviin sovelluksiin. (Cousins, 2019.)

Adobesta on saatavilla useita eri versioita hinnoitteluperusteisesti. Luonnollisesti hinnat määräytyvät sen mukaan millaiseen käyttöön sovellus tulee.

3.7.2 Postman

API-rajapintojen testaamiseen, kehittämiseen ja tutkimiseen tehty ohjelma. Suosittu työkalu, jolla voi helposti tutkia kehitysvaiheessa oman APIn toimivuutta turvallisessa ympäristössä. Kutsuja voi tehdä monella eri protokollalla useisiin kohteisiin kerralla ja tarkat sekä yksityiskohtaiset vastaukset kertovat selkeästi, jos jokin ei toimi tai toimii niin kuin on suunniteltu. Kutsuja voi käsin muokata juuri sellaisiksi kuin on tarve ja kaikki tarvittava löytyy vierekkäin välilehdiltä.

Postmanilla voi myös automatisoida API rajapintojen testausta, joka on erittäin tärkeää ja käytännöllistä. Kun ohjelmasta saadaan uusi versio aikaan ja se rakennetaan automatisoitua järjestelmää käyttäen, voidaan Postman suoraan lisätä siihen mukaan omana osanaan.

Postman on ilmainen avoimen lähdekoodin sovellus peruskäytössä. Se sopii yksittäiselle kehittäjälle tai kun kehitystiimi on pieni tai API rajapinnat yksinkertaisia. Suuremmille tiimeille tarjotaan yhteistä alustaa, jolla kaikki voi toimia samaan aikaan yhdessä, sekä lisää työkaluja, joita ei ilmaisessa versiossa ole. Näitä löytyy kolme eri tasoa, joista kahdessa ensimmäisessä hinta riippuu käyttäjien määrästä. Viimeisessä versiossa paketti räätälöidään tarpeiden mukaan ja myös hinta tarjotaan sen mukaan yrityskohtaisesti. (Postman inc.)

3.7.3 Visual Studio 2019

Microsoftin luoma kehitysympäristö, jolla voi luoda monenlaisia sovelluksia usealla eri kielellä. Web-sovellusten kehitys onnistuu alusta loppuun asti ja riippuen sovelluksen laajuudesta ja tarpeista sekä valituista teknologioista, vaadittavat komponentit ovat käytössä samassa ympäristössä. Pilvipalveluiden yleistyessä ja Microsoftin tarjotessa omaa Azure-ympäristöään aivan kaikkeen kehitykseen, on myös se suoraan yhdistetty Visual Studion rakenteisiin. Tämän ansiosta web-sovelluksen testaus, rakentaminen ja julkaisu onnistuu nappia painamalla.

Tietokantojen luominen, käyttäminen ja muokkaaminen onnistuu myös omilla tehokkailla työkaluilla. Käyttöliittymän ansioista SQL kieltä ei tarvitse välttämättä osata ollenkaan, vaan kehitysympäristö hoitaa sen taustalla. Mahdollisuus on myös käyttää pelkkää SQL, jos pätevyys siihen riittää ja se tuntuu luonnollisemmalta vaihtoehdolta. Yhdistäminen myös ulkoisiin tietokantoihin onnistuu erittäin helposti ja tuki löytyykin pelkistä tietokantatiedostoista ulkoisiin palvelimiin asti. Ja jos jotain tarvitsemaa ei löydy suoraan, voi tarvittavan paketin löytää Visual Studion Marketista, jonne kuka tahansa voi laittaa omia

lisäosiaan vapaaseen jakeluun kaikille käyttäjille. Myös oman voi luoda tarvittaessa ja mahdollisesti jakaa sen myös muille. Oma osionsa löytyy myös Microsoftin omalle Azure pilvipalvelulle, jonka yhdistäminen projektiin kuin projektiin on todella yksinkertaista ja intuitiivista.

Visual Studiosta on saatavilla kolme eri versiota. Yhteisöversio on ilmainen täysversio kehitysympäristöstä, johon on saatavissa kaikki samat ilmaiset laajennusosat kuin maksullisiinkin versioihin. Ilmainen on tarkoitettu yksittäisille kehittäjille, opiskelijoille ja harrastelijoille. Yritykset, joilla on isommat kehitystiimit, on tarjolla maksulliset versiot, joihin sisältyy Azure pilvipalvelun krediittejä, sekä ylimääräisiä Microsoftin kehittämiä työkaluja testaukseen ja diagnosointiin. Esimerkkinä kalleimmalle Enterprise versiolle annetaan, jos yrityksellä on enemmän kuin 250 tietokonetta tai yli miljoona Yhdysvaltain dollaria vuosittaista liikevaihtoa.

Visual Studio 2019 ei ole avoimen lähdekoodin sovellus. Sitä on pitkään kehitetty ja nimi muuttuu julkaisuvuoden vuosiluvun mukaan, kun suurempi versiopäivitys on saatavilla. (Microsoft)

3.7.4 SQL Server Management Studio

Kokonaisvaltainen ohjelma tietokantojen hallinnointiin. Samalla työkalulla voi hallita kaikkia osia SQL palvelimesta tai pilvessä olevasta Azure tietokannasta. Selkeät graafiset apuvälineet ja piirtotyökalut auttavat havainnollistamaan monimutkaisenkin kannan helpommin ymmärrettävänä visuaalisena tuotoksena. Monipuoliset skripti editorit antavat mahdollisuuden kaiken tasoisille kehittäjille ja järjestelmänvalvojille hoitaa tietokantaa kullekin omien kykyjensä mukaisesti.

Sisältää paikallisesti toimivan EXPRESS palvelin ohjelmiston, jolla voi kehitystä, testausta ja optimointia tehdä ilman erillistä palvelinta tai pilvipalvelua. Rajoitettu toiminnallisuudeltaan verrattuna SQL palvelimen täysiversioon. Tietokantojen koko sekä laitteistotuki on huomattavasti heikompi, tehden siitä vaikean käyttää täysivaltaisessa päivittäisessä käytössä.

Microsoftin luoma ja ilmaiseksi tarjoama kaikille käyttäjille. Myös kaupallisille yrityksille. Saatavilla myös usealla eri kielellä. (Microsoft)

3.7.5 Webstorm

JetBrainsin luoma kehitysympäristö web-sovelluksien hallintaan. JavaScriptiin pohjautuvat teknologiat onnistuvat samalla ohjelmalla säästäen näin aikaa ja vaivaa sekä lisäten

tehokkuutta. Kehitetty tukemaan suoraan tämän päivän yleisimpiä kieliä ja tyylejä. Node.js käyttö myös suoraan rakennettu ympäristöön, jonka käyttö toimii graafisesti hiirellä tai sisäänrakennetulla komentokehotteella.

Yksikkötestaus toimii suoraan samassa ohjelmassa ja sen voi liittää osaksi automaattista koodin tarkastusta ennen koodin rakentamista. Sovelluskehityksessä versionhallinta on erittäin tärkeä osa-alue ja myös sen Webstorm hoitaa sisäänrakennetuilla moduuleilla. Yleisimmät palvelut kuten Git ja Mercurial löytyy suoraan valikoista, joihin kirjautuminen on yksinkertaista ja niiden käyttö helppoa. (JetBrains s.r.o)

3.7.6 Azure ja Azure Devops

Microsoftin kehittämä ympäristö, joka yhdistää monia työkaluja ja alustoja yhteen. Pilvessä toimiva, johon saa tietokantoja ja internetsivuja pyörimään turvallisesti ja tehokkaasti. Microsoftilla on datakeskuksia ympärimaailmaa ja näistä voit valita missä sovelluksesi toimii. Siitä riippumatta tietysti on se saatavilla ympäri maailman.

Sisältää myös diagnostiikkatyökaluja, joilla näkee kävijämääristä datan latauksen nopeuteen ja mahdollisiin ongelmiin, jos käyttäjämäärät esimerkiksi kasvavat nopeasti.

Etuna on vahva rakenne pohjalla. Verrattuna oman palvelimen ylläpitoon, tietoturva on huipussaan ja laitteistoa on taustalla niin paljon, että toimintavarmuus on äärimmäisen korkea. Myös se, että henkilökuntaa ja teknikkoja on Microsoftilla töiden ääressä ympäri vuorokauden takaa, että ongelma tilanteet ratkaistaan mahdollisimman nopeasti.

Laskutus palvelulla toimii käytön mukaan, minimien ollessa erittäin alhaiset. Yksinkertaisesti luottokortilla voi kuka tahansa ottaa palveluita käyttöön.

Azuren DevOps tarjoaa monipuoliset työkalut jatkuvaan kehitykseen automaatio apuna. Kuten muutkin vastaavat palvelut se osaa hakea lähdekoodin ja jos se on muuttunut, päivittää sovellus ja ajaa kaikki automaatio testit ja muut tarvittavat tehtävät ennen julkaisua. Koska se on osa Azurea, diagnostiikka ja raportointi toimii suoraan samoilla työkaluilla samalta alustalta, joka tekee siitä nopean ja luotettavan.

Azure DevOps on pienimmillä resursseilla ilmainen, joka toimii erittäin hyvin pienillä tiimeillä ja projekteilla. Kalleimmillaan palvelu maksaa useita satoja dollareita kuukaudessa ja sillä saa kaiken mahdollisen käyttöön ja enemmän yhtäaikaisia töitä toimimaan rinnakkain. Lopullinen hinta määräytyy haluttujen resurssien määrän mukaan kuten Azuren peruspalveluissa.

3.7.7 Git ja GitHub

Git on versionhallinta järjestelmä, joka on kehitetty vuonna 2005 ja on tällä hetkellä maailman suosituin vaihtoehto versionhallintaan. Se toimii paikallisesti käyttäjän omalla koneella ja siihen voidaan tehdä merkintöjä sovelluksen kehityksen eri vaiheissa. Näin muutoksista jää näkyviin selkeä polku, johon voidaan myöhemmin aina palata, jos uusien versioiden kanssa tulee ongelmia tai halutaan jostain toisesta syystä palata kehityksessä taaksepäin.

Erikoisuutena on koodien haaroittaminen omiin osiinsa. Näin uutta ominaisuutta tehdessä, voidaan kyseinen koodi tallentaa omaan osioonsa, jotta ei vahingossa rikota jo toimivaa koodia. Näitä eri haaroja voidaan tarkastella, yhdistellä ja muokata tarpeiden mukaan vapaasti.

Git on kaikille käyttäjille ilmainen.

GitHub on pilvipalvelu, joka toimii samalla periaatteella kuin Git paikallisesti. Suositun GitHubista tekee sen mahdollisuus jakaa ohjelmointiprojekti helposti usean ihmisen kesken. Jokainen näkee lähdekoodin ja siitä tehdyt haarat. GitHub tarjoaa samat toiminnot kuin Git, mutta yksinkertaistaa sitä tuomalla siihen graafisen käyttöliittymän.

Erona näiden välillä on projektin mahdollinen julkinen osa. Git toimii paikallisesti vain itselleen, GitHubissa projekti voidaan laittaa julkisesti näkyviin, jolloin kuka tahansa näkee koodin ja riippuen valitusta lisenssistä, jatkaa kehitystä omana projektinaan tai tarjota omia ratkaisujaan projektin ylläpitäjälle. Tähän perustuu moni avoimen lähdekoodin projekti, jossa kuka tahansa voi osallistua kehitykseen.

GitHubia voi myös käyttää omalla työpöydällä erikseen ladattavalla sovelluksella, joka hoitaa samat asiat, mutta tarjoaa lisätyökaluja kehittyneimmille kehittäjille.

GitHub on maksullinen palvelu, mutta ilmaista perusversiota voi käyttää kuka tahansa, joka vain luo profiilin ja antaa henkilötietonsa. Opiskelijoille on tarjolla ilmaiseksi maksullinen täysiversio ja kaikki siihen liittyvä. (Devmountain)

4 Arkkitehtuurit

4.1 REST-rajapinta

REST edustaa sanoja Representational State Transfer ja se on vuonna 2000 esitelty, Roy Fieldingin kehittelemä, arkkitehtuurinen tyyli hajautettujen sovellusten välillä (Fielding, 2000).

API puolestaan edustaa sanoja Application Programming Interface. Termi kääntyy suomeksi lyhyesti sovellusrajapinta.

REST-rajapintoja käytetään siis sovelluksien väliseen kommunikointiin ja datan siirtoon tarkoitettu silta. Nykypäivänä REST -rajapinnat ovat niitä, jotka pitävät web- ja mobiilisovellukset toiminnassa, kuten Kivisaari toteaa blogissaan.

> *Jos tarkastellaan asiaa sovellustasolla niin vähissä ovat esimerkiksi mobiilisovellukset, joista ei ole yhteyksiä mihinkään ja kaikki tapahtuu paikallisessa sovelluksessa puhelimessa. Jos ja kun yhteyksiä on, ne ovat poikkeuksetta toteutettu REST API -rajapintoja hyödyntäen. (Kivisaari, 2016.)*

Kivisaari (2016) toteaa, että REST rakentuu hyvin vahvasti http:n päälle, sen luojan, Fieldingin, ollessa myös http-standardin luoja Tim Berners-Leen kanssa. Se, että REST rakentuu http-protokollan päälle tarkoittaa sitä, että se on maailman testatuin rajapinta http-siirtoprotokollan ohella.

REST-rajapinnasta puhuttaessa avainkäsitteenä informaatiolle on resurssi. Resurssi voi olla mikä tahansa informaatio, jota voidaan nimetä. Esimerkiksi se voi olla dokumentti, kuva tai väliaikainen palvelu. (restfulapi.net)

Kivisaari (2016) jatkaa REST:n toimintaperiaatteesta, siinä olevan kyse resursseista ja resurssien käytöstä http-metodien avulla, joita ovat GET, POST, DELETE, UPDATE ja PATCH, toisin sanoen CRUD-komennot, jotka ovat kätevästi myös tietokannan komennot.

GET sananmukaisesti tarkoittaa resurssien hakemista. Haku voi tapahtua ilman parametrejä, jolloin saadaan kaikki mahdolliset tai esimerkiksi nimen tai ID tunnisteen perusteella vain tietyt resurssit.

POST lähettää tietoa eteenpäin ja tätä käytetään esimerkiksi, kun lomakkeita internetsivuilla täytetään. Sen jälkeen tiedot lähetetään palvelimelle vahvistettavaksi ja lisättäväksi tietokantaan.

DELETE toiminnolla poistetaan resursseja. Sen kanssa täytyy olla tarkka ja käyttää yksilöivää tunnusta pyyntöä tehdessä, ettei vahingossa poisteta kaikkia tietoja.

UPDATE ja PATCH on tehty päivittämään olemassa olevia resursseja. Ero näiden välillä on, että update päivittää koko resurssin kerralla, eli jos haluat esimerkiksi, vaikka vain etunimen muuttaa, täytyy kaikki muutkin tiedot antaa samalla kertaa tai resurssissa on silloin tyhjiä kohtia, vaikka näin ei aikaisemmin ollut. Patch puolestaan toimii juuri päin vastoin, eli sillä voi päivittää vaikka yhden kentän tiedot kerralla. Sillä voi myös päivittää kaikki kerralla, jolloin se ei eroa updatesta millään tavalla.

REST-rajapinta ottaa vastaan XML- tai JSON-muotoista dataa ja välittää sitä tietokantaan tai toiseen suuntaan sovellukselle tai sovelluksesta sovellukseen. Datan lähettäminen ja hakeminen usein tapahtuu id-tunnisteen avulla.

5 Valmis sovellus

5.1 Sovelluksen rakenne

Opiskeluissa on vahvasti keskitytty full stack kehitykseen, joten tätä opinnäytetyötä varten sovellus on myös sillä periaatteella tehty. Sillä tarkoitetaan palvelinohjelmiston eli back endin ja käyttäjän näkemän sovelluksen eli front endin yhdistymistä toimivaksi kokonaisuudeksi. Lisäksi käytössä on tietokantoja, joiden avulla pystytään tallentamaan tietoja sessioiden välissä pysyvästi ja huolehtimaan turvallisesta käyttäjätietojen hallinnasta. Myös sovellukseen kirjautuminen hoituu tietokannan avulla.

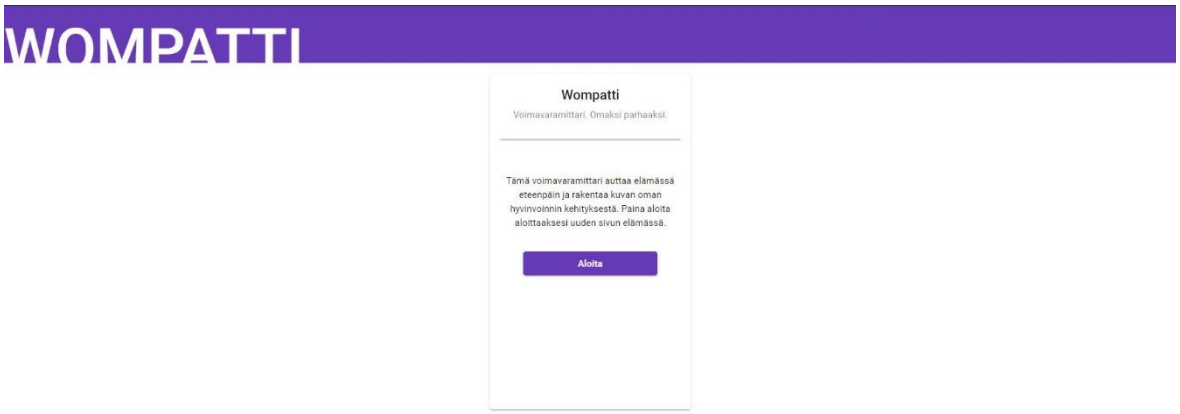
5.1.1 Web-applikaatio

Sovelluksen front end päätettiin toteuttaa Angularilla. Tästä on eniten kokemusta ja se on yksi suosituimmista kehityskehyksistä tällä hetkellä, joten dokumentaatiota ja apua ongelmakohtiin on saatavilla runsaasti. Angular toimii Single-Page-Application periaatteella, eli selain vaihtaa reaaliajassa sivun sisältöä navigaation tai http-kutsujen mukaan. Sivua ei ladata uudestaan jokaista linkkiä tai nappulaa painettaessa, jolloin käyttäjälle jää sulava ja viiveetön kokemus. Jokainen osa on erillinen komponentti, joita voidaan ladata samaan aikaan useita samalla ruudulle tai yksinkertaisella navigaatio komponentilla muuttaa sisältöä komponentista toiseen. Näin noudatetaan hyviä ohjelmointitapoja edelleen ja jokainen komponentti hoitaa vain oman tehtävänsä ja tämä lisää koodin selkeyttä, sekä luettavuutta.

Angularin taustan toiminta perustuu palveluihin, joista jokainen hoitaa saman periaatteen mukaisesti vain sen oman tehtävänsä. Siinä missä käyttäjä näkee komponenttien sisällön ruudullaan, palvelut toimivat taustalla piilossa käyttäjältä. Sovelluksen kaikki palvelut hoitavat asynkronisia http-kutsuja eri kohteisiin. Ensimmäisenä käytetään kirjautumiseen luotua palvelua, jonka tehtävä on lähettää tunnus ja salasana palvelimelle, jotta voidaan vahvistaa sallittu käyttäjä. Tämän jälkeen käyttäjälle avataan etusivu, joka samalla hakee taustalla aihealueet ja piirtää ne selkeästi ja yksinkertaisesti ympyrään siistejä kuvakkeita käyttäen. Eri alueille siirryttäessä haetaan tarvittavat resurssit ja ne tallennetaan paikallisesti, jotta niiden käyttö olisi nopeaa ja tarvittavat haut yksinkertaisempia. Näin tehdään vähemmän turhia kutsuja taustajärjestelmään ja nopeutetaan sovelluksen toimintaa.

Valmis Angular-sovellus pitää sisällään tervetuloa-sivun (Kuva 1), josta käy ilmi minkälaista tuotetta käyttäjä operoi. Päätimme pitää erillisenä sivuna kirjautumisen ja tervetuloa-sivun, koska tällä tavoin käyttäjä pakotetaan keskittymään siihen mitä ruudulla näkyy, eikä vain

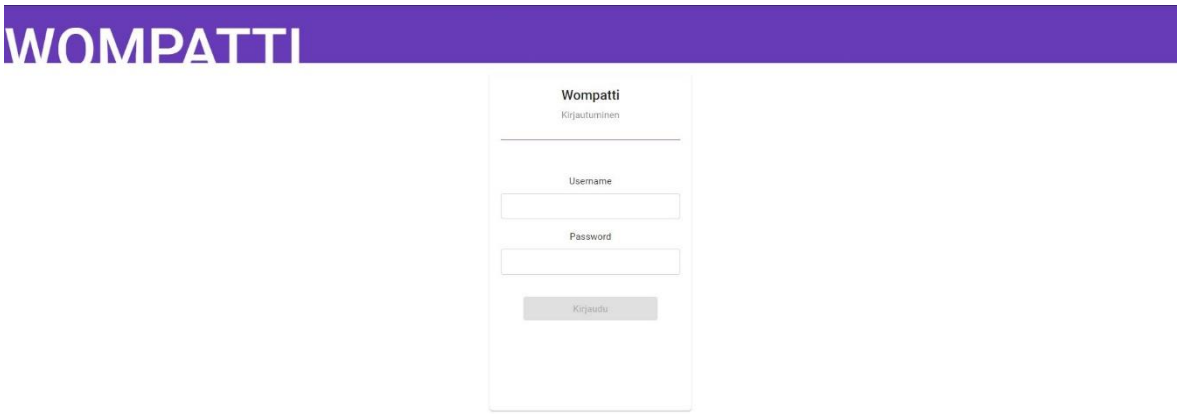
aloita kirjautumista sokkona. Sovelluksen kohderyhmä ja käyttötarkoitus vaativat käyttäjän ymmärtävän minkälainen sovellus on kyseessä.



Kuva 1. Tervetuloa

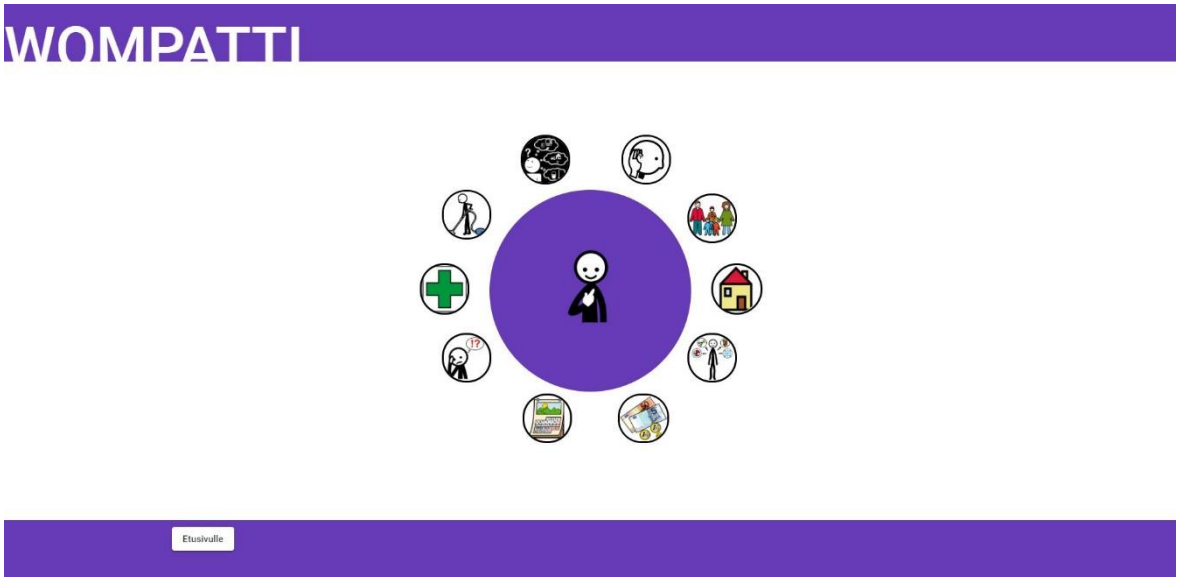
Aloita-painiketta painamalla pääsee kirjautumissivulle (Kuva 2), joka on rakennettu Angular Materialin form-moduulilla. Sovellus käyttää JSON Web Tokenia huolehtimaan kirjautumisen tilasta ja jos JWT on vanhentunut, ohjaa sovellus kirjautumislomakkeelle automaattisesti. JWT mahdollistaa myös automaattisen uudistamisen, jos käyttö jatkuu sen voimassaoloaikana. Tätä ominaisuutta ei kuitenkaan otettu käyttöön, koska sitä ei vaatimusmäärittelyissä merkitty tärkeäksi tai

tarpeelliseksi.



Kuva 2. Kirjautuminen sovellukseen

Kirjautumisen jälkeen käyttäjä ohjataan päävalikkoon (Kuva 3), josta löytyy helpon havainnollistamisen vuoksi ympyrämuodossa aihealueet. Aihealueet ovat näin tasa-arvoisessa asemassa toisiinsa nähden. Tasa-arvoisuus tässä tapauksessa on tärkeää, koska esimerkiksi aihealueena Raha ei voi olla tärkeämpi kuin vaikka Perhe ja Läheiset. Tasa-arvoinen asettelu myös ohjaa käyttäjää vastaamaan kaikkiin aihealueisiin. Listattu valikko voi johtaa ylikuormituksen tunteeseen pitkänä lista näkymänä. Päävalikossa on myös ohjeet Angular Materialin dialog-moduulilla aihealueiden käyttöön. Sovellus pyrkii olemaan aina valmiina tarjoamaan ohjeita ja apua käyttäjälle.

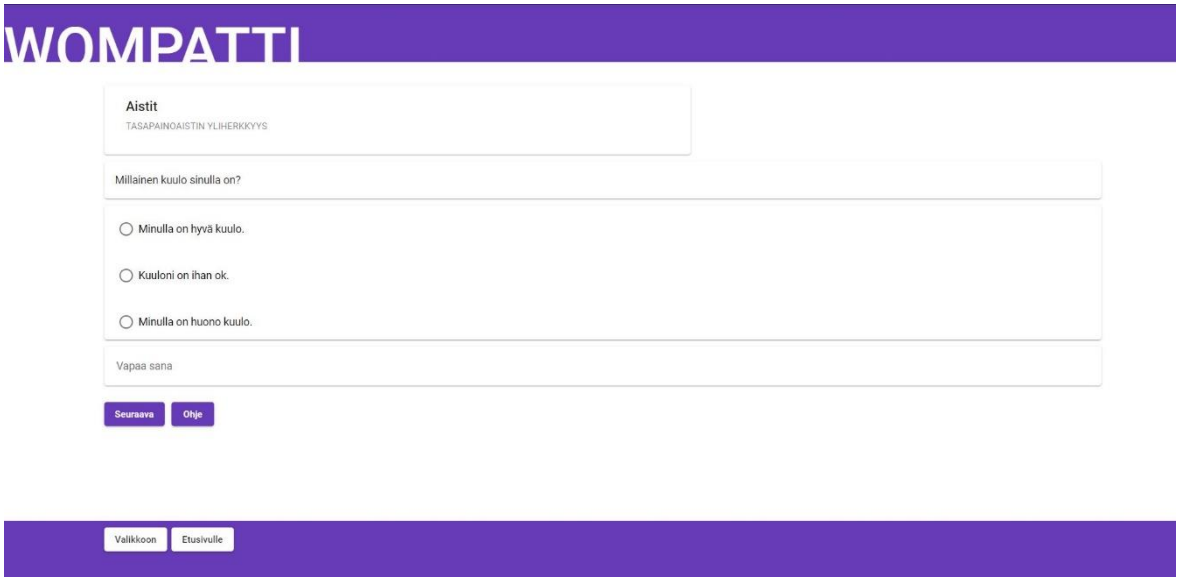


Kuva 3. Päävalikko ja aihealueet

Kuvassa 4 näkyy miten aihealueet aukeavat valikosta klikkaamalla yksi kerrallaan. Aihealueiden aktivointi tapahtuu URL-parametreilla, jotka kuljettavat aihealueen id-tunnisteen päävalikosta aihealuesivulle (Kuva 4). Id:n avulla haetaan tietokannasta http-protokollan GET-kutsulla vastaavan alueen Id-tunnus ja silmukassa näitä verrataan keskenään sekä luodaan kyseisen aihealueen kysymyksistä uusi taulukko. Taulukkoa ajetaan silmukassa Seuraava-painikkeen avulla läpi, jokaisen kysymyksen välissä.

Seuraava-painikkeen funktiossa tapahtuu paljon asioita samaan aikaan, mutta Angularin kevyen rakenteen ansiosta se ei tee painikkeen toiminnasta raskasta. Painikkeella ohjataan sivulle uusi kysymys taulukosta, luodaan vastausvaihtoehdot omaan Angular Materialin mat-card-moduuliin, tallennetaan edellinen vastaus ja vapaa-sana kentän tiedot uuteen taulukkoon sessiovarastoon sekä lähetetään sama vastaus http-protokollan POST-komennolla tietokantaan. Lisäksi painikkeessa on toiminto, joka tyhjentää valinnat ja vapaa-

sanakentän.



Kuva 4. Kyselysivu

Tietokantaan kysymykset tallentuvat niin, että samaa istuntoa on mahdollista jatkaa toisella kirjautumiskerralla. Saman istunnon kysymyksille luodaan yhteinen id-tunniste, jolla voidaan aikaleiman kanssa hakea edellinen istunto ja jatkaa mittariin vastaamista. Samalla periaatteella kysymykset noudetaan myös tulossivulle. Tuloksissa on mahdollista tarkastella vastauksia pylväskaaviossa, joka on tehty analyysityökaluselvityksestä valitulla kolmannen osapuolen chart.js-paketilla. Tuloksia voidaan suodattaa hakemalla päivämäärä ja/tai aihealue. Saman aihealueen tuloksia on myös mahdollista vertailla keskenään eri päivämäärien mukaan. Valitut tulokset ja muodostetut kaaviot on mahdollista tallentaa tai tulostaa PDF-formaattiin.

Kesken kyselyn tai tulosten tarkastelun on myös mahdollista navigoida pohjapalkin navigointipainikkeiden avulla päävalikkoon, kirjautumissivulle tai tuloksiin. Käyttäjän mukaan järjestelmänvalvojan tunnuksilla navigointipainikkeissa näkyy myös Työkalut-painike, jolla voidaan navigoida työkalusivulle. Järjestelmänvalvojalle löytyy työkalut käyttäjien lisäämiseen, aihealueiden tietojen muokkaamiseen, kysymysten lisäämiseen ja muokkaamiseen sekä käyttöoikeuksien jakamiseen.

5.1.2 REST-rajapinta

API aloitettiin .NET Core 3.0 versiolla, mutta nopeasti koko projekti jouduttiin päivittämään 3.1 versioon, koska uusimmat versiot lisämoduuleista eivät enää toimineet vanhemmalla versiolla. Rakenne jaoteltiin hyvien käytäntöjen mukaisesti eri tasoille, jotta jokainen sovelluksen osa olisi mahdollisimman yksinkertainen ja helposti ymmärrettävä.

Controller tasolla http kutsu otetaan vastaan ja ohjataan oikealle servicelle. Service tasolla tehdään tarvittavat tarkistukset ja mahdolliset muutokset kutsuihin ja sen jälkeen, kun on varmistettu kutsun oikea oppinen muoto, lähetetään se Repository tasolle. Tässä vaiheessa tehtävä on vain lukea tai kirjoittaa tietokantaan haluttu tehtävä ja palauttaa tehty kutsu sellaisenaan takaisin. Näin tiedetään takaisin tulleessa vastauksessa, että kaikki on mennyt oikein eikä virheitä sattunut matkalla.

Kirjautumista lukuun ottamatta, kaikki päätepisteet eli endpointit ovat suojattuja ja näin ollen vaativat hyväksytyn kirjautumisen päästäkseen käsiksi resursseihin. Vain kirjautuminen on avoin, koska muuten et voi kirjautua, kirjautuminen vaatisi kirjautumisen. Uusien käyttäjien luominen myös vaatii kirjautumisen, koska tarkoitus on hallita sallittujen henkilöiden pääsyä sovellukseen.

Rajapinta toimii kirjautumista lukuun ottamatta synkronisesti, jolloin ohjelma jää odottamaan pyyntöjen vastausta ennen suorituksen jatkamista. Valinta johtuu enemmänkin tähän asti opittujen menetelmien luonteesta ja asynkronisten järjestelmien vähäisestä kokemuksesta C# kielellä. Suurta vaikutusta suorituskyvyssä ei näy, koska synkroninen rajapinta on otettu huomioon web sovelluksessa.

5.1.3 Tietokannat

Sovelluksessa on kaksi SQL relaatiotietokantaa. Ensimmäinen on automaattisesti luotu käyttäjien hallintaan varten Microsoftin ASP.NET Core Identity moduulilla. Se on API, joka hoitaa käyttäjät, salasanat, roolit ja monet muut käyttäjien todentamiseen tarvittavat asiat. Tukee myös ulkopuolista kirjautumista (external login provider), joihin kuuluu Facebook, Google, Microsoft tili sekä Twitter. Identity moduulin lähdekoodi on saatavilla julkisesti GitHubista. Tämä tietokanta on se, joka hoitaa sovellukseen kirjautumisen ja huolehtii käyttäjille määritellyistä oikeuksista.

Toinen tietokanta on varsinainen sovelluksen käyttämä, johon on tallennettu aihealueet, kysymykset, vastaukset ja toinen lista käyttäjistä. Koska tässä yhteydessä hyvinvoinnin seurantaa tehdään ohjaajan kanssa avustetusti, kysymyksiin vastaajat eivät itse kirjaudu järjestelmään. Tämä luo erillisen, paljon yksinkertaisemman taulun vastaajista, johon ei tarvita turhaan tietoja salasanoista, sähköpostiosoitteista tai sallituista oikeuksista. Näin pystytään kuitenkin pitämään kirjaa käyttäjien vastauksista ja silti pitämään järjestelmä mahdollisimman yksinkertaisena.

5.1.4 Azure ja Azure Devops

Azureen tietokannan, rajapinnan ja web-sovelluksen julkaisu ovat yksinkertaisia prosesseja. Tietokannan julkaiseminen tapahtuu ensimmäisenä uuden yhteysmerkkijonon saamiseksi. Merkkijonon avulla rajapinnan julkaisu onnistuu Azureen tietokantayhteys säilyttäen. Rajapinnan julkaisu tapahtuu kätevästi Visual Studion sisäänrakennetun julkaisutyökalun ansiosta. Työkalu julkaisee rajapinnan suoraan Azureen sille luodulle palvelulle.

Web-sovelluksen julkaiseminen vaatii Azureen muodostettavan App Servicen, jonka avulla luodaan sivulle domain. Web-sovelluksen julkaisemisessa on tärkeää myös muistaa muuttaa environment.ts tiedostojen asetukset koodieditorissa.

Azure Devopsiin työn vieminen onnistuu parhaiten versionhallinnan integroinnilla Devopsiin. Lisäksi Azure Devopsiin pitää luoda jatkuvan toimittamisen putket, joiden avulla julkaisu saadaan automatisoitua yhden napin painalluksen päähän.

Ongelmana Azure Devopsiin viemisessä muodostuu käyttöoikeudet. Versionhallinnan toteutus oli tässä työssä virheellistä. Rajapinnan, tietokannan sekä web-sovelluksen on syytä olla perustettu tietovarastoja versionhallintaan yhden käyttäjän alle. Ongelmana tässä tapauksessa oli se, ettei käyttöoikeudet jakamalla täydet käyttöoikeudet siirry käyttäjätunnukselta toiselle, vaan ainoastaan push, pull ja merge oikeudet eli niin sanotut perusoikeudet tietovarastojen käyttöön. Nyt tehdyissä julkaisuissa yhden käyttäjän alta ei omatoimisesti onnistunut asettaa toisen käyttäjätunnuksen alle luotua tietovarastoa jatkuvan toimittamisen putkeen.

5.2 Työn ongelmat ja niiden ratkaisut

JSON Web Tokenin uudistus käytön aikana olisi ollut testausta helpottava ominaisuus, varsinkin kun kirjautumisen voimassaoloaika oli määritelty testausvaiheessa lyhyeksi. Toisaalta tällä tavalla kirjautumisen toiminnan testaus tuli hoidettua huolellisesti ja useaan kertaan.

Työssä tuli vastaan useita ongelmia liittyen tietokannan rakenteen ja sovelluksessa tarvittavien tietojen välillä. Ongelmia olisi voitu välttää tietokannan perusteellisemmalla suunnittelulla. Suunnitteluun käytetty aika luultiin riittäväksi, mutta todellisuus todisti toisin. Muutokset, joita tietokantaan tehtiin, olivat helposti muokattavissa lisäämällä tauluihin avaimia sekä niille parametrejä. REST-rajapinnassa muutokset hoituivat nopeasti EF Core Power Toolsien Reverse Engineering-toiminnolla, kun tietokantayhteys oli valmiiksi luotu.

REST-rajapinnassa oli ongelmina satunnaiset kirjoitusvirheet, joista ei kuitenkaan tullut huomautuksia koodia kirjoittaessa. Vasta rajapintaan yhteyttä ottaessa ongelmat ilmenivät, mutta hoituivat helposti. Lisäksi rajapinnan käynnistystiedostossa oli moduulit väärässä järjestyksessä. Myös CORS eli Cross-Origin-Resource-Sharing oli muodostettu väärin, jolloin ongelma ilmeni vasta web-sovelluksella dataa hakiessa. Ongelma poistui koodia uudelleen oikein kirjoittamalla.

Web-sovelluksessa ilmenneet ongelmat olivat korrelaatiossa rajapinnan ongelmiin. Http-kutsujen rakentamisen yhteydessä tuli ilmi tietokannan sekä rajapinnan ongelmat.

Tämän takia voimme päätellä, että full stack sovellusta tehdessä ongelmat eivät useinkaan johdu ainoastaan yhdestä pinon komponentista, jota rakennetaan, vaan ongelmat, joita ilmenee juontavat juurensa muihin komponentteihin.

6 Johtopäätökset

Johtopäätöksissä pyrimme pohtimaan sovelluksen tekoprosessia, ohjelmoijien yhteistyötä, työtapoja ja vastaamaan kysymyksiin kuten miten tähän päädyttiin ja miten alkuperäiseen verrattuna sovellus on kehittynyt sekä mikä on sosiaali- ja terveydenhuollon tilanne tällä hetkellä sekä julkisella että yksityisellä puolella.

6.1 Sovelluksen nykytila

Full stack-sovelluksen rakentaminen on yksinkertainen prosessi aiheen oppineelle ja työvälineet tuntevalle. Tekoprosessissa korostuu perusteellinen suunnittelu ja huolellinen toteutus. Saimme huomata, että alustava työ sovellusta varten edesauttaa ongelmien välttämistä koodia kirjoittaessa ja mahdollistaa sovelluksen loppuun saattamisen vähemmällä vaivalla ja vastoinkäymisillä. Erityisesti tietokannan ja REST-rajapinnan toteutus luo stabiilin pohjan web-sovelluksen rakentamiselle.

Vaatimusmäärittelyt, joita ennen sovelluksen ohjelmointia pohdimme ja kirjasimme, olisi hyvä käydä läpi ja pohtia tietokannan ja rajapinnan toteutuksen kannalta. Suurin osa vaatimusmäärittelyistä koskivat web-sovellusta, mutta vaikuttivat merkittävästi edellä mainittujen komponenttien rakenteisiin. Emme ottaneet näitä huomioon ja web-sovelluksen koodia kirjoittaessa huomasimme tietokannan ja rajapinnan puutteet.

Kuten aiemmin mainittua, päädyimme työn nykyiseen tilaan erinäisistä haasteista huolimatta ja olemme tyytyväisiä tulokseen. Emme saaneet viivästymisen vuoksi kaikkia komponentteja tehtyä valmiiksi asti ja koodiin varmasti jäi myös virheitä, joita emme huomanneet. Vaatimusmäärittelyistä välttämättömimmän prioriteettiasteen ominaisuudet kuitenkin pystyttiin pääsääntöisesti toteuttamaan loppuun asti.

Johtopäätöksenä voisimme todeta, että erittäin perusteellisella ja huolellisella suunnittelulla voidaan alustaa sovelluksen valmistumisen mahdollisuuksia huomattavasti.

6.2 Dokumentointi ja työpäiväkirjat

Työn perusteellinen dokumentoiminen on ohjelmointiprojektissa tärkeässä asemassa. Dokumentoiminen tuotannossa olevassa työssä auttaa virheenhallinnassa, jatkokehityksessä sekä koodin lukemisessa, esimerkiksi tiimin uuden työntekijän näkökulmasta. Työmme dokumentointi toteutettiin koodia kommentoimalla suoraan niin sanottuun tuotantoversioon. Koimme tämän toistaiseksi hyvänä vaihtoehtona tämän kaltaisessa projektissa, jonka ei ole tarkoitus olla kaupalliseen tarkoitukseen tai muuhun julkiseen käyttöön. Tiedostamme silti tämän huonot puolet valmiissa tuotteessa. Koodista

tulee meluisaa ja epämiellyttävä lukea, mikäli jokaista riviä on kommentoitu toiminnan selittämisellä. Oikein dokumentoituna tuotannossa oleva koodi on puhdasta ja selkeää hyvien ohjelmointitapojen mukaisesti, jolloin sama koodi on kommentoituna ja toimintaperiaatteet pohjustettuna erillisenä tallenteena.

Kokemuksemme työpäiväkirjan pitämisestä on hyvä silloin kun se sisältää kattavasti työtapojen sekä tehdyn työn kerronnan. Työpäiväkirjan ansiosta työskentelyä voi seurata ja se auttaa hallitsemaan omaa ajankäyttöä. Työpäiväkirjan hyviin puoliin lukeutuu myös ohjelmoijan työtapojen helpompi kehittäminen. Omaa työpäiväkirjaa analysoimalla pystyy pohtimaan millä tavalla samankaltaisia työvaiheita pystyy jouduttamaan. Jälkeenpäin tarkastellessa pystytään myös seuraamaan omaa ajankäyttöä ja kuinka työskentely onnistuu etätyönä, kuten tässä tapauksessa jouduimme tekemään.

Työpäiväkirjojen pitäminen koettiin yleisesti hyväksi tavaksi toimia. Päiväkirjat muodostuivat kolumneista, jotka pitivät sisällään tehdyn päivämäärän ja työskentelyajat, yksityiskohtaisen selvityksen päivän työstä sekä avainsanoja, joiden avulla päiväkirjoja pystyy suodattamaan.

Työpäiväkirjoja tutkimalla huomaa tekijöiden asettuvan kahteen eri profiiliin päivärytmillisesti. Toinen tekijöistä suoritti työnsä niin sanottuina toimistoaikoina, maanantaista perjantaihin. Toisella taas päivämäärällä ja kellonajalla ei ollut niinkään suurta merkitystä. Työpäivän pituus oli kutakuinkin sama. Pääasiallisena avainsanana nousee esille Angular ja front end. Tämä ei sinänsä yllätä, koska front end oli sovelluksen laajin osa vieden suurimman prosenttiosuuden käytetystä työajasta.

Toisena merkittävänä huomiona työpäiväkirjoista voidaan todeta tekijöiden motivaation laskevan perjantaihin mennessä, sen ollessa tehottomin päivä valmistuvien komponenttien määrässä mitattuna. Kyseinen huomio voi myös tämän kokoluokan projektissa olla sattumaakin.

Tiimissä tai parityöskentelyssä prosessin tärkeimpään asemaan nousevat aikataulujen noudattaminen, huolellinen yhteissuunnittelu sekä avoin ja selkeä kommunikaatio. Uskoisimme työn viivästyneen edelleen, mikäli yhteistyö olisi ollut matalammalla tasolla. Kommunikaation merkitys nousi kyseisen työn toteutuksen luonteen vuoksi. Vallitsevan maailman tilanteen ansiosta meillä ei ollut mahdollisuutta työskennellä samassa työpisteessä, joten jouduimme tekemään työn etänä. Tiivis yhteistyö mahdollisti sovelluksen näinkin nopean toteuttamisen.

6.3 Aikataulutus ja yhteistyö

Tässä kappaleessa käydään läpi aikataulutusta ja sen toimivuutta sekä haasteita, joita aikataulutus aiheutti työn aikana. Lisäksi luvussa käydään läpi tekijöiden yhteistyötä.

Kehitys viivästyi suunnitellusta aikataulusta noin kuukaudella. Joukko sairastumisia ja vastoinkäymisiä hidasti sovelluksen rakentamista merkittävästi. Työlle asetettiin välietappeja, jotka pääosin pystyttiin saavuttamaan sovitussa aikataulussa. Työtä tehtiin niin sanottuina sprintteinä kahden viikon jaksoissa. Kahden viikon jälkeen keskusteltiin uusista välitavoitteista, joita kohti työ seuraavassa sprintissä suunnataan. Aikataulutusta muokkasivat useaan otteeseen muun muassa tietokannan suunnittelu sekä REST-rajapinnan rakenne, joiden muutostyöt olivat sovelluksen toiminnan kannalta kriittisiä.

Aikataulusta viivästyminen tarkoitti myös sitä, että osa tulosten tarkastelun toiminnoista, jouduttiin karsimaan. Tavoitteena oli pystyä hakemaan tuloksia useammilla parametreillä kuin mitä nykyinen sovellus tekee. Tämänhetkiset hakuparametrit ovat riittävät mittariin vastanneelle henkilölle. Mittarin vastaajalle ei tarvitse osoittaa muunlaisia kaavioita, kuin päivämäärä sekä aihealue. Järjestelmänvalvojalle voitaisiin osoittaa useampia eri mahdollisuuksia vastausten hakemiseen.

Viivästyminen johti myös siihen, ettei tekijöillä ollut mahdollisuutta kirjoittaa koodia täysin yhdenmukaiseksi. Pikaisen aikataulun myötä koodiin jouduttiin tekemään soveltavia ratkaisuja eikä täten koodi mukaile jokaisen toiminnon osalta samaa yleisilmettä.

Lisäksi aikataulun kiristyminen aiheutti työn osa-alueiden karsintaa. Esimerkiksi testitapausten ohjelmoiminen ei ollut mahdollista. Azure Devopsiin saatiin tämän takia ainoastaan koodin automaattinen rakennus sekä julkaisu. Automaattiset testitapaukset jäivät tässä tapauksessa vain tavoitteiden tasalle.

Tekijöiden välinen yhteistyö hoitui koronaviruksen aiheuttamasta kaoottisuudesta huolimatta sujuvasti etätyönä. Tekijöiden välinen kommunikaatio oli päivittäistä ja ongelmatilanteita puitiin ja ratkaisuja etsittiin usein yhdessä.

6.4 Erot mittareiden välillä

Työ eroaa aiemmasta hankkeelle tehdystä sovelluksesta ulkonäöllisesti ja toiminnallisesti. Perustoiminnot ovat pääosin samat erilaisella ohjelmoinnillisella toteutuksella. Aiempi työ ei käsittänyt tietokantaa eikä REST-rajapintaa, jotka tästä työstä löytyvät. Tuoreempi sovellus on näiden ansiosta huomattavasti monipuolisempi sekä helpommin sovellettavissa

muuhunkin mittaamiseen. Tietokannan rakenne mahdollistaa useamman eri kohderyhmän kysymysten säilyttämisen. Käytännön kokemusta tästä ei ole ja on potentiaalinen jatkokehityksen aihe. Lisäksi sovelluksessa on mahdollista kirjautua ja tehdä mittarin kysymyksiä useammassa sessiossa. Aiempi sovellus mahdollisti vain anonyymin lähestymistavan yhteen sessioon.

6.5 Sosiaali- ja terveydenhuollon tarve ja nykytila

Kokemuksemme aiemman hanketyön ja tarvekartoituksen pohjalta on, että tämänkaltaisille mittareille on julkisella puolella tarvetta. Olemme saaneet huomata, että merkittävä osa mittaamisesta ja arvioinneista toteutetaan julkisella puolella kynä ja paperi menetelmin, jolloin tallennus digitaaliseen muotoon tapahtuu skannaamalla ja pdf-tallennuksella.

Ennen opinnäytetyöprosessia teimme tarvekartoitusta mittarin tarpeesta, ja siitä kuka tämänkaltaista mittaria tarvitsee. Huomasimme, että mittarille on tarvetta niin kentällä, kuin tutkimustyössäkin. Vastauksena digitaalisen hyvinvointimittarin käyttöönotolle saimme lauseen "*Tällaisen mittarin tulisi lyödä ensin paperisena läpi, ennen kuin siitä otettaisiin käyttöön digitaalinen versio".* Mielestämme edeltävä lause kertoo paljon julkisen sosiaali- ja terveyshuollon tilasta digitalisaation suhteen sekä asenteista, joita digitaaliset ratkaisut voivat kohdata. Sovelluksen tuottaminen lähes valmiiksi otti kahdelta opiskelijalta kokonaisuudessaan noin kuukauden ja sovellus skaalautuu usean eri kohderyhmän tarpeiden mittaamiseen. Sovelluksen ylläpidollinen hinta on pieni verrattuna fyysisten tulosteiden ja tarvikkeiden hankkimiseen sekä työajan säästymiseen.

Kokemuksemme mukaan yksityinen sektori on tässä asiassa jo valovuosia edellä julkista sektoria.

7 Jatkokehitys

Potentiaalia jatkokehitykselle tämän kaltaisessa sovelluksessa on huomattavasti. Kehityksen kohteena näkisimme hyvinvointimittarin tuloksien tarkastelun laajentamisen useampiin eri parametreihin. Esimerkiksi järjestelmänvalvojalle voitaisiin osoittaa oikeudet hakea useampi käyttäjä kerrallaan ja muodostaa kaavioita suuremmasta massasta dataa. Näin sovellusta voisi kehittää enemmän tutkimustyölle suunnattuihin toimintoihin.

Jatkokehitykselle mahdollisia alueita ovat koodin selkeyttäminen ja uudelleen kirjoittaminen yhdenmukaiseksi. Lisäksi virheenhallinnan lisääminen koodiin on jatkokehityksessä välttämätöntä. Kaikkiin http-protokollan kanssa toimiviin funktioihin tarvitaan virheenhallintaa. Virheenhallinnan alle sijoittuu myöskin sovelluksen mahdolliset umpikujat, jotka vaativat sivua ei löydy-komponentin.

Tietokannan kehittäminen useammalle eri kohderyhmän kysymyksille. Tämä on nykyisessä rakenteessa mahdollista toteuttaa, mutta käytännön kokemus puuttuu. Kyseinen toimenpide vaatisi tietokannan rakenteen uudelleen arvioinnin.

Jatkokehitys vaatii oikean tilaajan sovellukselle, jotta tarvittavia ominaisuuksia ja käytännöllisyyttä pystyttäisiin räätälöimään tarkemmin. Käyttäjäkohtaisia kokemuksia myös vaaditaan enemmän, koska kehittäjä näkee sovelluksen eri tavalla ja ymmärtää miten sovelluksen pitäisi toimia verrattuna siihen mitä oikeasti tapahtuu.

Lähteet

Google, 2020. Introduction to Angular Concepts. Google. Oppaat. Viitattu 09.10.2020. Saatavissa https://angular.io/guide/architecture

Google, 2020. CLI Overview and Command Reference. Google. Oppaat. Viitattu 9.10.2020. Saatavissa https://angular.io/cli

Joyent. About Node.js. Joyent. Viitattu 9.10.2020. Saatavissa https://nodejs.org/en/about/

Chand, M. 2020. What is C#. C# Corner. Artikkelit. Viitattu 21.10.2020. Saatavissa https://www.c-sharpcorner.com/article/what-is-c-sharp/

ISO 9075 1987 Information processing systems - Database language – SQL. International Organization for Standardization. Viitattu 3.11.2020. Saatavissa https://www.iso.org/standard/16661.html

IBM. What is relational database. IBM. Viitattu 3.11.2020. Saatavissa https://www.ibm.com/analytics/relational-database

Microsoft. Introduction to the C# language and .NET. Microsoft. Oppaat. Viitattu 3.11.2020. Saatavissa https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/

Microsoft. What is .NET. Microsoft. Oppaat. Viitattu 3.11.2020. Saatavissa https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

npm Inc, 2020. About npm. npm Inc. Oppaat. Viitattu 3.11.2020. Saatavissa https://docs.npmjs.com/about-npm

npm Inc, 2020. About the public npm registry. npm Inc. Oppaat. Viitattu 3.11.2020. Saatavissa https://docs.npmjs.com/about-the-public-npm-registry

Adobe, 2020. What is XD. Adobe. Oppaat. Viitattu 4.11.2020. Saatavissa https://helpx.adobe.com/fi/xd/how-to/what-is-xd.html

Cousins, C. 2019. What is Adobe XD? a 101 Intro. Adobe. Artikkelit. Viitattu 4.11.2020. Saatavissa https://designshack.net/articles/software/what-is-adobe-xd/

Kivisaari, T. 2016. API:t ovat modernin integraatiostrategian ydin. Digia. Blogit. Viitattu 6.11.2020. Saatavissa https://blog.digia.com/rest-api

Fielding, R. 2000. Architectural Styles and the Design of Network-based Software Architectures. Kalifornian yliopisto. Väitöskirja. Viitattu 6.11.2020. Saatavissa https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Postman Inc. Viitattu 6.11.2020. Saatavissa https://www.postman.com/

restfulapi.net. 2020. What is REST. restfulapi.net. Viitattu 6.11.2020. Saatavissa https://restfulapi.net/

Microsoft. Viitattu 6.11.2020. Saatavissa https://visualstudio.microsoft.com/vs/

Microsoft. 2020. What is Microsoft SQL Server Management studio (SSMS). Microsoft. Oppaat. Viitattu 7.11.2020. Saatavissa https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15

JetBrains s.r.o. Viitattu 7.11.2020. Saatavissa https://www.jetbrains.com/webstorm/

Devmountain. 2020. Git vs. GitHub: What's the difference. Devmountain. Blogit. Viitattu 7.11.2020 https://blog.devmountain.com/git-vs-github-whats-the-difference/

Chartio. 2020. Quick Start. Chartio. Oppaat. Viitattu 14.5.2020 https://chartio.com/docs/quick-start/

Microsoft 2020. Power BI Embedded. Viitattu 14.5.2020 https://azure.microsoft.com/en-us/services/power-bi-embedded/#product-overview

Metabase 2020. Metabase Documentation. Oppaat. Viitattu 14.5.2020 https://www.metabase.com/docs/latest/

KNIME AG 2020. KNIME Documentation. Oppaat. Viitattu 14.5.2020 https://docs.knime.com/

Google 2020. What can you do with Data Studio. Oppaat. Viitattu 14.5.2020 https://support.google.com/datastudio/answer/6283323?hl=en#

Chart.js 2020. Introduction. Oppaat. Viitattu 4.11.2020 https://www.chartjs.org/docs/latest/