Verkkosivut psykoterapiatyön tukena

Milla Tuomainen

Haaga-Helia ammattikorkeakoulu
Tradenomi, tietojenkäsittely
Amk-opinnäytetyö
2024

Tiivistelmä

Tekijä
Milla Tuomainen

Tutkinto
Tradenomi, tietojenkäsittely

Raportin/Opinnäytetyön nimi
Verkkosivut psykoterapiatyön tukena

Sivu- ja liitesivumäärä
35

Tämä toiminnallinen opinnäytetyö käsittelee verkkosivujen suunnittelua ja toteutusta psykoterapeutille. Opinnäytetyössä pyritään vastaamaan ammattiryhmän tarpeisiin suunnittelemalla ja toteuttamalla verkkosivut, joilta asiakas löytää helposti ja nopeasti tarvitsemansa tiedon palveluntuottajasta ja hänen tarjoamastaan psykoterapiapalvelusta. Lisäksi verkkosivut pyritään toteuttamaan siten, että psykoterapeutti voisi itse jatkossa ylläpitää ja tarvittavin osin päivittää niitä myös vähäisillä tietoteknisillä perustaidoilla.

Opinnäytetyössä käsitellään verkkosivujen suunnittelussa huomioon otettavia tekijöitä sekä käytännön toteutusta. Tarkoituksena on, että lopputulos toimisi referenssinä osaamisestani sekä osia siitä voisin hyödyntää myöhemmin myös muihin tarpeisiin tehtävissä verkkosivuissa.

Toiminnallisen opinnäytetyön tuloksena syntyi helppokäyttöiset ja selkeät, responsiiviset verkkosivut. Verkkosivujen päivittäminen tekstisisällön osalta oli mahdollistettu ylläpitäjän oikeuksilla toimivalle henkilölle tietokantatoimintoja hyödyntäen yksinkertaisen käyttöliittymän kautta.

Asiasanat
WWW-sivut, Verkkosivujen suunnittelu, Ohjelmointi, Psykoterapia

Sisällys

1 Johdanto

Tämä toiminnallinen opinnäytetyö käsittelee verkkosivujen suunnittelua ja toteutusta psykoterapeutille. Psykoterapeuttien rooli mielenterveyspalveluissa on merkittävä ja psykoterapeuteista on suuri pula suhteessa tarvitsijoihin. Mielenterveyspalvelujen toimivuuden parantamiseksi on käynnissä ja suunnitteilla monia toimia, joilla pyritään tuomaan psykoterapian tarvitsijat matalalla kynnyksellä lähemmäs palveluita. (Kuusimäki 5.1.2023; Venermo 21.2.2022) Perinteiset asiointitavat vähenevät myös mielenterveyden palveluiden osalta ja nettiterapiat, omahoito-ohjelmat ja asiointipalvelut yleistyvät. Tätä taustaa vasten peilaten yrittäjänä toimivan psykoterapeutin palvelujen tuominen helpommin lähestyttäväksi potentiaaliselle asiakkaalle vaatii markkinoinnin, tiedottamisen ja yhteydenoton mahdollisuuden netin kautta. Samaan aikaan psykoterapeutin digiosaaminen saattaa olla perustaitojen ulkopuolella monesti vajavaista.

Voidaan siis päätellä, että psykoterapeuttien vähäisen määrän ja toisaalta suuren tarpeen vuoksi on tarve pyrkiä saattamaan asiakkaat ja palveluntarjoajat mahdollisimman tehokkaasti yhteen nettiä hyödyntäen. Tietotekninen lisäkoulutus on varmasti mielenterveyspalvelujen tarjoajille tarpeen, mutta mahdollisuuksien mukaan heidän arvokas työpanoksensa olisi tarpeen suunnata asiakastyöhön. Opinnäytetyössä pyritään vastaamaan edellä kuvailtuihin tarpeisiin suunnittelemalla ja toteuttamalla verkkosivut, joilta asiakas löytää helposti ja nopeasti tarvitsemansa tiedon palveluntuottajasta ja hänen tarjoamastaan psykoterapiapalvelusta ja, joita psykoterapeutti voisi itse jatkossa ylläpitää myös vähäisillä tietoteknisillä perustaidoilla.

Ensin opinnäytetyössä käsitellään verkkosivujen suunnittelua teoreettisesta näkökulmasta ja esitellään eri osa-alueita, joita suunnittelussa tulee ottaa huomioon. Verkkosivujen toteutus perustuu huolella laadittuun suunnitelmaan, jonka lähtökohtana ovat asiakkaan ja loppukäyttäjän tarpeet. Visuaalisen ilmeen suunnittelu on ensimmäinen askel ja halutun ulkoasun löytyessä, sitä hiotaan haluttuun toimivaan muottiin ottaen huomioon saavutettavuus ja responsiivisuus eri laitteille. (Beaird, Walker & George 2020, luku 1)

Verkkosivujen teknisen toteutuksen osalta esitellään eri ohjelmointikieliä ja niiden käyttötarkoituksia sekä verkkosivujen toteutukseen tarvittavia muita työkaluja ja huomioon otettavia asioita. Verkkosivujen rakentamisen tekninen kenttä on niin laaja, ettei sitä voitu kattavasti kuvailla tämän opinnäytetyön laajuus huomioiden. Näin ollen esittely rajattiin pääasiassa opinnäytetyössä käytettyihin teknologioihin sekä ohjelmointikieliin.

Opinnäytetyön toisella osuudella tutustutaan käytännön tasolla siihen, miten verkkosivut suunniteltiin sekä millaisiin ratkaisuihin päädyttiin ja miksi. Koska asiakas ja loppukäyttäjä ovat tässä opinnäytetyössä teoreettiset, suunnitelmaluonnos lopullisista sivuista muotoutuu tarkastelemalla

psykoterapeuttien tarpeita ammattikuntana ja tulevan verkkosivun käyttäjän eli potentiaalisen tulevan asiakkaan näkökulmaa tutkimalla.

Verkkosivujen toteutuksen osalta kuvaillaan palvelinpuolen toimintojen sekä käyttöliittymän toimintojen rakentamista vaihe vaiheelta. Toimintojen rakentaminen aloitettiin palvelinpuolen toiminnallisuudesta, joka ei ole suoraan näkyvissä käyttäjälle. Se huolehtii esimerkiksi tietokantayhteyksistä, mitkä mahdollistavat käyttöliittymän toiminnot. Tämän jälkeen siirryttiin rakentamaan sitä osaa verkkosivuista, jonka käyttäjä näkee ja jonka kanssa vuorovaikutus tapahtuu. Nämä kaksi osuutta on erotettu eri osioihin selkeyttämään lukijalle opinnäytetyön toteutuksen etenemistä.

Lopuksi pohditaan valmiin lopputuotoksen eli julkaisuvalmiin verkkosivuston onnistumista sekä sitä, miltä osin se työskentelyn päätteeksi vastasi aluksi tehtyä suunnitelmaa. Pohdinnassa otetaan huomioon opinnäytetyön tavoitteet ja arvioidaan niiden toteutumista. Tässä osiossa arvioidaan myös kriittisesti oman toteutuksen käytännön sovellutuksia ja jatkokäyttöä.

2 Verkkosivujen suunnittelu

2.1 Visuaalinen ilme

Useat tutkimukset ovat osoittaneet, että visuaalisen ilmeen perusteella tehty arvio verkkosivusta luodaan jopa alle kymmenesosasekunnissa ja se on melko pysyvä mielikuva riippumatta sivuston myöhemmin luetusta sisällöstä. Lisäksi suurin osa käyttäjistä klikkaa pois epämieluisalta sivulta jo muutaman sekunnin selaamisen jälkeen. (Selejan ym. 2016) Tämä tieto nostaa visuaalisen suunnittelun erittäin merkittävään rooliin. Suunnittelussa tulee ottaa huomioon käyttäjän tarve löytää tarvitsemansa tieto tai toiminto verkkosivuilta nopeasti sekä luoda loppukäyttäjää esteettisesti miellyttävä ulkoasu (Beaird, Walker & George 2020, luku 1).

Käytettävyyttä palvelevat esimerkiksi selkeästi löytyvä navigaatio, jonka linkit on nimetty kuvaavasti ja yhtenäinen tyyli kaikilla sivuston osilla (Beaird, Walker & George 2020, luku 1). Merkittävässä roolissa käytettävyyteen ja verkkosivun ilmeeseen ovat värien käyttö ja erityisesti valitut väriyhdistelmät, typografia ja kuvat. Esteettisesti miellyttävän tyylin löytäminen vaatii sekä yleisesti graafisen suunnittelun että kohderyhmän tuntemusta. Graafisen suunnittelun periaatteiden huomioimisesta on hyötyä verkkosivuilla sisällöstä riippumatta. Esimerkiksi symmetrian sekä kultaisen leikkauksen suhteiden hyödyntäminen sivun osasten asettelussa on järkevää. Haluttua sanomaa voi korostaa visuaalisin keinoin leikittelemällä kontrastilla, etäisyyksillä, mittasuhteilla, toistolla ja sijoittelulla. (Beaird, Walker & George 2020, luku 1)

Kohderyhmän hahmottamisen ja mahdollisesti lisäksi kävijäpersoonan määrittelyn avulla on helpompi ymmärtää, minkälainen sisältö ja tyyli puhuttelisi juuri tämän verkkosivun käyttäjiä. Kun visuaalinen ilme tukee kohdeyleisölle suunnattua, tavoiteltua brändin mukaista mielikuvaa, tämä yhtenäisyys ja johdonmukaisuus tyylissä tuo kävijöille selkeyden ja turvallisuuden tunnetta. (Karjalainen 22.9.2021) Nämä tuntemukset puolestaan kasvattavat sivuilla vietettyä aikaa ja johtavat todennäköisemmin toimintakehotusten toteutumiseen. Toimintakehotus voi olla painike tai linkki, joka ohjaa kävijää esimerkiksi ostopolulla eteenpäin. (Venermo 21.2.2022)

2.2 Responsiivisuus

Valittu visuaalinen ilme on tärkeä saada toimimaan kauniisti eri kokoisilla näytöillä. 2000-luvun alussa ohjaaminen erilliselle mobiilisivulle oli vielä toimivaa, mutta nykypäivänä näyttöjen monimuotoisuus on ohjannut suunnittelun ”Mobile First” lähestymistapaan. Siinä suunnitellaan ensin verkkosivun asettelu yksinkertaiseksi, nopeasti latautuvaksi ja mobiiliystävälliseksi. Vasta sen jälkeen lisätään raskaampia sivuelementtejä, kun selaimen olosuhteet sen sallivat. Tämä on järkevää

paitsi näyttökoon näkökulmasta myös mobiililaitteiden heikomman muistin, suorituskyvyn ja mahdollisesti hitaampien verkkoyhteyksien vuoksi. (Beaird, Walker & George 2020, luku 1)

2.3 Saavutettavuus

Saavutettavuuden ajatus on pyrkiä varmistamaan, että mahdollisimman laaja joukko ihmisiä voi vaivattomasti hyödyntää verkkopalveluja. Se edellyttää moninaisuuden ja erilaisuuden huomioimista verkkopalvelujen suunnittelussa ja toteutuksessa. (Aluehallintovirasto s.a. a) Suunnittelussa huomioitavat erilaiset käyttäjät voivat olla esimerkiksi ruudunlukijoiden tai puheohjauksen käyttäjiä, heikkonäköisiä, kuulovammaisia tai käyttäjiä, joilla on autisminkirjo, lukivaikeuksia tai motorisia rajoitteita (Aluehallintovirasto s.a. b). Saavutettava verkkosivu on teknisesti hyvin toteutettu, helppokäyttöinen ja ymmärrettävä (Aluehallintovirasto s.a. c).

Teknisesti hyvin toteutetun sivuston tunnistaa siitä, että sen lähdekoodi on virheetöntä ja loogisesti rakennettua, koodaamisessa on huomioitu responsiivisuuden tarpeet eri laitteille sekä verkkosivut toimivat myös avustavilla teknologioilla. Huomioitavia asioita ovat esimerkiksi erilaisille laitteille ja laitteen asennon mukaan skaalautuva asettelu, kuvien vaihtoehtoiset kuvaukset (”alt”) ja videoiden tekstivastineet, HTML-elementtien käyttäminen sisällön hierarkian merkitsemiseksi sekä looginen lukusuunta lähdekoodissa. (Aluehallintovirasto s.a. c; Österlund 28.2.2022)

Onnistuneella visuaalisella ilmeellä; värien ja kuvien käytöllä, typografian valinnalla ja asettelulla voidaan luoda helppokäyttöinen sivu. Näillä kaikilla keinoilla saadaan aikaiseksi hierarkia, jotka on helppo seurata ja löytää tarvittavat toiminnot. Esimerkiksi riittävät suuret ja väljät toimintopainikkeet ja lomakkeet, aiheen ymmärtämistä tukevat kuvat ja videot, johdonmukaisesti ryhmitellyt osiot sekä riittävät kontrastit väreissä tukevat verkkosivun haluaman sisällön välittämistä myös käyttäjälle, jolla on erityistarpeita. (Aluehallintovirasto s.a. c; Österlund, 25.10.2022)

Ymmärrettävän sisällön tuottaminen verkkosivulle vaatii mm. selkeän kielen käyttöä, tekstin jäsentämistä helppolukuisiksi riittävän lyhyiksi kokonaisuuksiksi sekä sisällön tarjoamista monikanavaisesti myös videoina, kuvina ja äänenä. (Aluehallintovirasto s.a. c; Österlund 28.2.2022) Suunnitelmallinen kohdeyleisön tarpeita vastaava sisällöntuotanto tukee parhaimmillaan yrityksen tavoitteita ja esimerkiksi lisää myyntiä (Kuusimäki 5.1.2023).

2.4 Verkkosivujen merkitys psykoterapiatyön tukena

Ammatinharjoittajana toimivan psykoterapeutin verkkosivut ovat tärkeä osa hänen henkilöbrändiään. Luottamus henkilöön ja tekemiseen, positiivinen mielikuva henkilöstä persoonana sekä ajatus vahvasta ammattitaidosta ja osaamisesta ovat onnistuneen henkilöbrändin merkkejä. Henkilöbrändin rakentaminen vie paljon aikaa ja rakentuu monesta tekijästä, mutta verkkosivut toimivat

parhaimmillaan käyntikorttina, jotka ovat tukevat brändiä johdonmukaisesti ilmeen ja sisällön osalta. (Koivunen 14.9.2018; Vahtola 2020, luku 13)

Henkilöbrändin tukemisen osalta tulee miettiä, minkälaisia asioita haluaa nostaa esille (Vahtola 2020, luku 13). Psykoterapeutin henkilöbrändin osalta voisi ajatella tärkeiksi asioiksi nousevan ammattitaidon ja osaamisen osoittavat näytöt, kuten koulutus, työkokemus ja erityiset osaamisalueet sekä näyttö aktiivisesta pyrkimyksestä kehittää omaa osaamistaan ja tietouttaan alan ollessa nopeasti muuttuva. Vahvan ammattiosaamisen lisäksi merkittävä tekijä psykoterapeutin henkilöbrändissä ovat potentiaalisessa asiakkaassa heräävät mielikuvat ammatinharjoittajan persoonasta ja työskentelytavasta. Olisi tärkeä osata tuoda esille omia työskentelytapojaan esimerkiksi kuvailun ja esimerkkien sekä mahdollisesti persoonaa ilmentävien kuvien kautta.

Psykoterapeutin sivut on kohdennettu nykyisille ja tuleville asiakkaille. Verkkosivut voivat toimia osana markkinointia esimerkiksi tarjoamalla tietoa palveluista, nostamalla näkyvyyttä hakukoneissa ja tarjoamalla helpon tavan ottaa yhteyttä ja siten johtaa asiakkuuteen. Nykyisille asiakkaille verkkosivut voivat tarjota erilaisia toimintoja, jotka ylläpitävät syntynyttä asiakkuutta ja tuovat kilpailuedun muihin vastaavaa palvelua tarjoaviin nähden. Verkkosivut voivat toimia myös itse ammatinharjoittajan työkaluna tarjoten analytiikkaa asiakkaista ja heidän käyttäytymisestään sekä toimintoja hallinnoida omaa työtään esimerkiksi ajanvaraustoimintojen osalta.

3 Verkkosivujen tekninen toteutus

Käyttäjälle näkyvästä verkkosivusta eli joukosta toimintoja, jotka tapahtuvat selaimessa voidaan käyttää nimitystä *frontend.* Frontendin päällimmäinen taso on *HTML* (HyperText Markup Language), joka pitää sisällään muotoilemattoman verkkosivun sisällön. Seuraava taso on *CSS* (Cascading Style Sheets), joka toteuttaa verkkosivujen muotoilun, kuten asettelun, värit ja taustat. Verkkosivujen toiminnot toteuttava taso on *JavaScript*, joka tällä hetkellä yleisimmin toteutetaan käyttäen React-kirjastoa. Tämä taso toimii myös linkkinä taustalla serverillä toimiville *backend*iksi kutsuttaville verkkosivun osille, kuten tietokantaan. (Nicoara 2023, luku 1) Backendin ohjelmointikielinä voidaan käyttää niin ikään JavaScriptia tai vaihtoehtoisesti esimerkiksi Pythonia tai Javaa. Palvelimella toimii monesti myös tietokanta, joka toimii muistina sovellukselle mahdollistaen monet toiminnot.

3.1 HTML ja CSS

HTML on standardoitu tietokonekieli, jota käytetään verkkosivujen ja -sovellusten luomiseen. HTML määrittää tekstin, kuvien, taulukoiden, linkkien ym. median sijoittamisen verkkosivuille käyttämällä yksinkertaisia tekstikuvauksia. Kun vierailet verkkosivustolla, palautetaan HTML-sivu. Selain ymmärtää HTML:ssä määritellyt sivun tiedot ja näyttää sen verkkosivuna. (Roy, Daniel & Agrawal 2023, luku 12; Nicoara 2023, luku 3)

CSS on tyylikieli, joka määrittelee, miten HTML-elementit näkyvät verkkosivulla. CSS koostuu säännöistä, jotka liittyvät eri HTML-elementteihin ja määrittelevät niiden tyylin, kuten värit, fontit, reunukset, välimatkat ja animaatiot. CSS:n käyttö mahdollistaa sivujen yhtenäisen tyylin määrittelyn ja ylläpitämisen helposti, kun visuaalisia muutoksia tehdään yhdestä paikasta moneen elementtiin. (Roy, Daniel & Agrawal 2023, luku 12; Nicoara 2023, luku 3)

CSS:n käyttö nopeuttaa ja helpottaa HTML-elementtien muotoilua, mutta vaatii joka tapauksessa jokaisen elementtityypin muotoilun erikseen. Valmiita CSS-kirjastoja käyttämällä voi prosessia nopeuttaa entisestään. Bootstrap-kirjasto on tällä hetkellä maailman suosituin työkalu responsiivisten verkkosivujen toteuttamiseen (Camus 5.5.2022). Se sisältää valmiiksi määriteltyjä CSS tyylejä ja komponentteja, joita voi käyttää suoraan HTML-tasolta. Lisäksi Bootstrapia voi muuttaa yksittäisen projektin tarpeiden mukaiseksi lisäämällä omia CSS-määrittelyjä ohittamaan Bootstrapin määrittelyt, muokkaamalla Bootstrap-tiedostoja tai hyödyntämällä Sass-kieltä (Syntactically Awesome Style Sheets). (Camus 5.5.2022)

3.2 Kuvankäsittely

Kuvankäsittelyohjelmat ovat tietokoneohjelmia, jotka mahdollistavat digitaalisten kuvien muokkaamisen ja manipuloinnin. Suosittuja kuvankäsittelyohjelmia ovat esimerkiksi Adoben tarjoamat Photoshop ja Illustrator, GIMP (GNU Image Manipulation Program) ja Corel PaintShop Pro.

Kuvilla ja grafiikoilla on suuri merkitys verkkosivujen toteutuksessa. Ne auttavat luomaan ensivaikutelman sivuista, tekevät verkkosivuista houkuttelevampia ja kiinnostavampia kävijöille, yhdenmukaistavat sekä välittävät brändin tunnelmaa ja haluttua viestiä. Parhaimmillaan visuaalinen sisältö selkeyttää navigaatiota ja tekee sisällöstä helpommin hahmotettavaa. (Beaird, Walker & George 2020, luku 1)

Kuvien merkitys käyttäjäkokemukselle liittyy myös siihen, että liian suuret kuvat saattavat hidastaa sivun latausaikoja ja heikentävät näin käyttäjäkokemusta ja hakukoneoptimointia. Ongelmaksi muodostuu se, että kuvien tulisi kuitenkin olla riittävän suuria, jotta ne näyttävät teräviltä ja selkeiltä isommilla näytöillä. Kuvat, jotka on optimoitu yhdelle näytölle, eivät välttämättä skaalaudu tai näytä hyvältä eri kokoisilla näytöillä. Yksi yleinen virhe on venyttää samaa kuvaa eri kokoisille näytöille, mikä voi johtaa epätarkkuuteen ja heikkoon visuaaliseen laatuun. Kuvankäsittelyn avulla kuvat voidaan optimoida niin, että ne säilyttävät laatunsa, mutta eivät liiallisesti hidasta sivuston latausaikoja. (Beaird, Walker & George 2020, luku 1)

SVG (Scalable Vector Graphics) -tiedosto on vektoriformaatti, joka tallentaa kuvan matematiikan ja geometrian avulla eikä pikseleinä, kuten JPEG ja PNG. Tämän takia kuvatiedostot ovat pienikokoisia verrattuna rasterigrafiikkaan ja lisäksi kuva skaalautuu eri näyttökokojen mukaan ilman laadun heikkenemistä. SVG-muotoisia kuvia käyttämällä vältytään muokkaamasta jokaisesta kuvasta sopivan kokoista versiota neljälle tai viidelle eri näyttökoolle. (Beaird, Walker & George 2020, luku 1)

3.3 JavaScript ja TypeScript

JavaScript on ohjelmointikieli, joka on johdettu Java-ohjelmointikielestä. Verkkosivujen kehityksessä se mahdollistaa dynaamisen sisällön ja interaktiivisuuden sivustolla. JavaScript suoritetaan selaimessa, mikä mahdollistaa sen vuorovaikutuksen käyttäjän kanssa ilman tarvetta palvelinpuolen koodille. JavaScriptin avulla voidaan lisätä monenlaisia toiminnallisuuksia verkkosivulle, kuten muuttaa sivun sisältöä dynaamisesti, hallita käyttäjän klikkauksia, lähettää ja vastaanottaa tietoa palvelimelta ilman sivun uudelleenlataamista (AJAX). (Roy, Daniel & Agrawal 2023, luku 12; Nicoara 2023, luku 4)

TypeScript on Anders Hejlsbergin suunnittelema avoimen lähdekoodin ohjelmointikieli, joka tehostaa JavaScriptin ominaisuuksia ja on täysin yhteensopiva JavaScriptin kanssa. Toisin sanoen

TypeScript on pohjimmiltaan JavaScript lisäominaisuuksilla, jotka tekevät koodin kirjoittamisesta ja ylläpidosta helpompaa ja vähemmän virhealtista. TypeScriptin avulla voi lisätä tyypit muuttujille, funktioille, parametreille ja paluuarvoille ja usein TypeScript pystyy päättelemään tyypit automaattisesti, mikä helpottaa koodin kirjoittamista. TypeScript tukee oliopohjaista ohjelmointia edistyneillä käsitteillä, kuten luokilla, rajapinnoilla ja perinnällä, mikä helpottaa erityisesti suurissa projekteissa. Lisäksi staattinen analyysi tarkistaa koodin virheiden varalta ennen sen suorittamista. (Nicoara 2023, luku 5; Roldán 2023, luku 2)

JavaScript-kirjastot ja -kehykset ovat valmiita koodikokoelmia, jotka on kehitetty ratkaisemaan yleisiä ohjelmointiongelmia tai tarjoamaan ennalta kirjoitettuja toimintoja nopeuttamaan koodaamista (McEven 16.2.2021). Kehys on kirjastoa laajempi kokonaisuus, joka määrittelee sovelluksen rakenteen ja ohjaa sen suunnittelua. Kirjastojen/kehysten valinta riippuu projektin tarpeista ja web-kehittäjän mieltymyksistä. (Camus 5.5.2022) Tänä päivänä on yleistä rakentaa myös backend käyttäen JavaScriptiä ja erilaiset kehykset ovat siinä suurena apuna (Roy, Daniel & Agrawal 2023, luku 12). Esimerkkejä suosituista JavaScript-kehyksistä ovat Angular, Vue.js ja Express.js. Angular on Googlen kehittämä laajojen ja monimutkaisten sovellusten rakentamiseen sopiva kehys. Vue on erityisen suosittu yksisivuisten sovellusten (single-page application) käyttöliittymien kehityksessä. Se on helppo ottaa käyttöön ja integroida muihin projekteihin. Express.js on Node.js-pohjainen backend web-sovelluskehys, joka tekee palvelinpuolen ohjelmoinnista JavaScriptillä helpompaa ja tehokkaampaa. (Khan 15.3.2021)

JavaScript-kirjastoja ovat muun muassa jQuery, React ja Axios. jQuery oli ensimmäisiä js-kirjastoja helpottamaan DOM-käsittelyä, tapahtumien hallintaa, animaatioita ja AJAX-kutsuja (Khan 15.3.2021). Axios on HTTP-asiakaskirjasto, jota käytetään tekemään HTTP-pyyntöjä. Sitä voidaan käyttää yhdessä minkä tahansa JavaScriptin kirjaston tai kehyksen kanssa. (Axios s.a.) React on kirjasto käyttöliittymäkomponenttien rakentamiseen. Se mahdollistaa tehokkaan ja dynaamisen käyttöliittymän rakentamisen yhdistämällä JavaScriptin ja HTML:n ominaisuudet.

3.4 Python

Maailman suosituimpiin ohjelmointikieliin lukeutuva Python on yksinkertainen ja tehokas korkean tason ohjelmointikieli. Avoimen lähdekoodin ohjelmointikielenä laajan yhteisön osallistuminen sen kehittämiseen vahvistaa sen asemaa entisestään. Python on luokkapohjainen oliokieli, mutta se tarjoaa mahdollisuuden myös proseduraaliseen ohjelmointiin ja funktionaaliseen ohjelmointiin. Python-kehittäjät voivat hyödyntää useampia eri ohjelmointiparadigmoja jopa saman ohjelman sisällä ja näin valita kullekin yksittäiselle ongelmalle tehokkaimman ratkaisun. (Holden, Martelli, Martelli Ravenscroft & McGuire 2023, luku 4)

Pythonin loogisessa ja helposti ymmärrettävässä syntaksissa sisennykset ovat tärkeässä roolissa, sillä lohkojen merkitsemiseen ei käytetä aaltosulkeita tai muita erottimia, kuten monessa muussa ohjelmointikielessä. Muuttujia voidaan määritellä suoraan ilman erillistä esittelyä ja niiden tietotyyppi määräytyy dynaamisesti suorituksen aikana. (Holden ym. 2023, luku 3)

Pythonin laaja standardikirjasto ja lisämoduulit sekä suuri määrä kolmannen osapuolen kirjastoja mahdollistavat sen soveltamisen laajasti erilaisiin käyttötarkoituksiin, kuten palvelinpuolen web-kehitykseen, datan monimutkaiseen numeeriseen käsittelyyn, tietokantaoperaatioihin, tekoälyyn ja pelinkehitykseen. Standardikirjaston moduulien ja työkalujen käyttäminen onnistuu ilman erillisiä asennuksia tai latauksia, ja helposti paketinhallintatyökalujen (esim. pip) avulla asennettavat Python-yhteisön luomat kirjastot täydentävät standardikirjastoa kattaen lähes kaikki ohjelmoinnin alueet. Pythonille on saatavilla esimerkiksi valmiita rajapintoja tietokantaohjelmistojen käyttöä varten. (W3 Schools s.a., Holden ym. 2023, luku 7)

3.5 Tietokanta

Tietokannat ovat keskeinen osa verkkosivujen toteuttamista, tarjoten tehokkaan tavan tallentaa, hakea ja hallita tietoa. Verkkosivun taustalla toimiva tietokanta mahdollistaa dynaamiset toiminnot sekä käyttäjien tietojen säilyttämisen, kuten käyttäjätilit ja ajanvaraustiedot. Tietokannat voivat olla SQL-relaatiotietokantoja, kuten MySQL tai PostgreSQL, tai NoSQL-tietokantoja, kuten MongoDB.

Relaatiotietokannat perustuvat relaatioteoriaan, joka tarjoaa rakenteen tietokantojen hallinnalle. Tällaisissa tietokannoissa tieto on järjestetty taulukkomuotoon riveiksi ja sarakkeiksi. Jokaisella taulukon rivillä on yksilöllinen tunniste, ja sarakkeet edustavat eri ominaisuuksia tai tietotyyppejä. Relaatiotietokantoja käytetään usein, kun tieto on tarkasti strukturoitua ja määriteltyä ja myös tietojen välinen suhde on tarkasti määritelty. Relaatiotietokannoista moni käyttää tiedon hakemiseen ja käsittelyyn SQL-kieltä (Structured Query Language), mikä on standardoitu tapa suorittaa kyselyitä tietokannasta. (Date 2019, luku 1; Ardeleanu 2016, luku 2)

NoSQL-tietokannat poikkeavat relaatiotietokannoista siten, että ne eivät noudata perinteisen relaatiotietokannan taulukkomallia, vaan esimerkiksi avain-arvopareja tai dokumentteja tiedon tallentamiseen. NoSQL-tietokannat tarjoavat joustavuutta ja helpon uudelleenjärjestelyn mahdollisuuden tilanteissa, joissa tietomallit voivat muuttua usein. Asiakirjapohjainen tallennus mahdollistaa esimerkiksi MongoDB:n kohdalla tiedon mallinnuksen JSON-formaatissa, joka konvertoidaan binääriseksi tietokantaan tallennusta varten. (Bierer 2020, luku 1)

3.6 Käyttäjävarmennus

Käyttäjän varmennus takaa sivuston turvallisuuden ja käyttäjädatan suojaamisen. Tämä varmistaa, että käyttäjät ovat oikeutettuja pääsemään tietyille sivustoalueille tai suorittamaan tiettyjä toimintoja. Tyypillisesti käyttäjän varmennus edellyttää, että käyttäjä tunnistautuu antamalla pätevät tunnistetiedot, kuten käyttäjätunnuksen ja salasanan. Käyttäjävarmennuksessa käytetään istuntokohtaisia tunnisteita tai evästeitä, jotka auttavat säilyttämään käyttäjän tilan selaimessa.

Varmennusmekanismit voivat vaihdella riippuen sivuston vaatimuksista ja turvallisuustarpeista. Yleisiä menetelmiä ovat käyttäjätunnus-salasana-yhdistelmät, kaksivaiheinen varmennus ja esimerkiksi Google-tilin avulla kirjautuminen (engl. social login). Käyttäjien tunnistustietoja, kuten salasanoja, tallennetaan usein tietokantaan salattuina ja turvattuina. Kaksivaiheinen varmennus on tehokas lisäkerros, joka vaatii käyttäjältä toisen todennusvaiheen, kuten tekstiviestin koodin tai mobiilisovelluksen generoiman koodin syöttämisen. Hyvin toteutettu käyttäjän varmennus vähentää merkittävästi riskiä väärinkäytöksille ja parantaa sivuston kokonaisturvallisuutta.

Verkkosivut käyttävät yleensä turvallisia salauksia varmennustietojen suojaamiseksi, ja tietokantoihin tallennetaan käyttäjien salatut salasanat. Web-sovelluskehykset, kuten Express.js, Django tai Ruby on Rails, tarjoavat usein sisäänrakennetun tuen käyttäjävarmennukselle. Lisäksi käyttäjäroolien hallinta on tärkeä osa varmennusta, jotta oikeudet voidaan määrittää tarkasti eri käyttäjäryhmille.

3.7 API-rajapinnat

API (Application Programming Interface) on rajapinta, joka mahdollistaa eri ohjelmistokomponenttien kommunikoinnin. Web-sovellusten parissa tänä päivänä suosituin tapa toteuttaa API on käyttää REST (representative state transfer) -arkkitehtuurimallia. Muita arkkitehtuurimalleja ovat esimerkiksi SOAP ja WSDL. RESTin periaatteita noudattavaa APIa voi kutsua RESTful API:ksi ja se tarjoaa yksinkertaisen tavan tietojen siirtoon palvelimelta web-sovellukseen. (Rodriguez 8.2.2015)

REST-suunnitteluperiaate käyttää http-metodeja resurssien hallintaan ja tiedon siirtoon: GET-metodia resurssin hakuun, POST-metodia resurssin luomiseen, PUT-metodia resurssin tilan muuttamiseen tai sen päivittämiseen ja DELETE-metodia resurssin poistamiseksi. Näitä neljää perustoimintoa kutsutaan CRUD-toiminnoiksi: akronyymi sanoista **c**reate, **r**ead, **u**pdate ja **d**elete. (Rodriguez 8.2.2015)

4 Toiminnallinen osuus: Suunnittelu

4.1 Tarkoitus, tarpeet ja lähtökohdat

Tämän toiminnallisen työn lähtökohtana on luoda verkossa toimivat sivut psykoterapeutin tarpeisiin sopien. Tarkoituksena on, että lopputulos toimisi referenssinä osaamisestani sekä osia siitä voisin hyödyntää myöhemmin myös muihin tarpeisiin tehtävissä verkkosivuissa. Valitsin juuri psykoterapeutin verkkosivut ammatin ollessa itselle tuttu oman toisen koulutukseni vuoksi, mutta sivut toimisivat mainiosti pienin muutoksin myös toisen asiakasrajapinnassa työskentelevän ammattilaisen käyttöön.

Verkkosivujen tarpeet kartoitettiin terapeutin työn sisältöä ja kohdekäyttäjän tarpeita ajatellen. Niiden tulisi sisältää vähintään toimiva navigaatio ja tietoa palveluista. Tärkeää on voida jakaa sisältöä näkyväksi käyttäjän roolin perusteella siten, että vierailijalla on pääsy sisältöön, mutta kirjautuneella ylläpitäjäkäyttäjällä olisi näkyvissä myös muokkausmahdollisuus tekstisisältöihin.

Uusien käyttäjien rekisteröityminen olisi mahdollista lisätä sivulle, mikäli myöhemmässä vaiheessa haluttaisiin lisätä esimerkiksi lomakkeita tai muuta kirjautuneelle käyttäjälle näkyvää sisältöä. Tässä vaiheessa sille ei ollut tarvetta, mutta toiminnallisuus päätettiin tehdä valmiiksi. Näin ollen rekisteröitymismahdollisuus olisi myöhemmin mahdollista joko asettaa vierailijoilta piiloon vain admin-roolissa kirjautuneen käyttäjän alaisuuteen tai vapaasti rekisteröitymispainikkeen alle kaikille vierailijoille näkyväksi.

4.2 Ulkoasu

Ulkoasun tulisi ilmentää terapeutin henkilöbrändiä. Koska tässä projektissa ei ole toimeksiantajaa, pyritään ulkoasun värimaailmalla, sommittelulla ja sisällöllä välittämään psykoterapialle sopivia mielikuvia, kuten rauhallisuus, seesteisyys ja lempeys. Terapeutin tulisi herättää asiakkaassa luottamusta asiantuntijuuteen, joten sisällön sävyn tulisi olla asiapitoinen. Lisäksi selkeys ja suoraviivaisuus yhdistetään asiantuntijuuteen koukeroisuuden ja ylitäyttämisen vastakohtana.

Opinnäytetyön tavoitteita ajatellen ulkoasun muokkautuvuus helposti esimerkiksi värejä vaihtamalla, on tärkeää. Näin ollen päädyttiin yksinkertaistettuun navigaatiopalkki/sisältö/alapalkki -rakenteeseen, jonka ulkonäköä pystyy muokkaamaan nopeasti fontteja ja värejä vaihtamalla ilman tarvetta kuvankäsittelyohjelmalla tehdyille muutoksille.

Kuva 1. Mood board, jossa valitut värit, fontit ja kuvitustyyli

Väreistä vihreä yhdistetään intuitiivisesti luontoon ja sitä kautta luonnon rauhoittaviin mielikuviin sekä kasvuun ja terveyteen. Ruskea on maanläheinen väri, joka sopii luontevasti yhteen vihreän kanssa ja herättää katsojassa mielikuvia turvallisuudesta ja lohdusta. Kuvaan 1 on kerätty verkkosivuille valitut vihreän ja ruskean sävyt. Suunnittelussa harkittiin myös sinisen ja liilan värejä, sillä myös ne olisivat voineet sopia herättelemään kohdeyleisössä tavoitteen mukaisia mielikuvia. Sininen väri herättää luontevasti mielikuvan taivaasta tai vedestä (mm. järvi tai meri). Se viestii harmoniaa, rauhaa ja tasapainoa. Toisaalta sininen saatetaan yhdistää myös suruun, alavireisyyteen ja kylmyyteen, mitkä eivät ole toivottuja mielikuvia. Sopivia mielikuvia saattaisi välittää myös liila väri, sillä se yhdistetään monesti viisauteen. Liila kuitenkin viestii herkästi myös vallan ja luksuksen mielikuvia ja värin liiallisen käytön varoitetaan saattavan herättää katsojassa turhautumisen tunteita. (Ferreira 2.11.2023)

4.3 Toiminnot ja ominaisuudet

Tarvittaviin toimintoihin päädyttiin analysoimalla asiakaspersoonan tarpeita. Sivujen yläosaan suunniteltiin navigointipalkki, jonka linkkejä klikkaamalla päästään eri näkymiin/sivuille. Navigaatiopalkin oikeasta reunasta löytyy mahdollisuus sisäänkirjautumiseen. Jotta verkkosivusta voisi hyödyntää osia mahdolliseen myöhempään käyttöön, suunniteltiin muutama hyödyllinen sivupohja tiedon jakamiseen, joita olisi helppo muokata tarpeen mukaisen sisällön esittelyyn esimerkiksi palveluista, ammatinharjoittajasta itsestään ja yhteydenotosta. Sivupohjiksi valikoituivat henkilöesittely-sivu, yhteydenottolomake-sivu ja kaksi erilaista tietosivua. Toisella tietosivulla rakenteen saisi

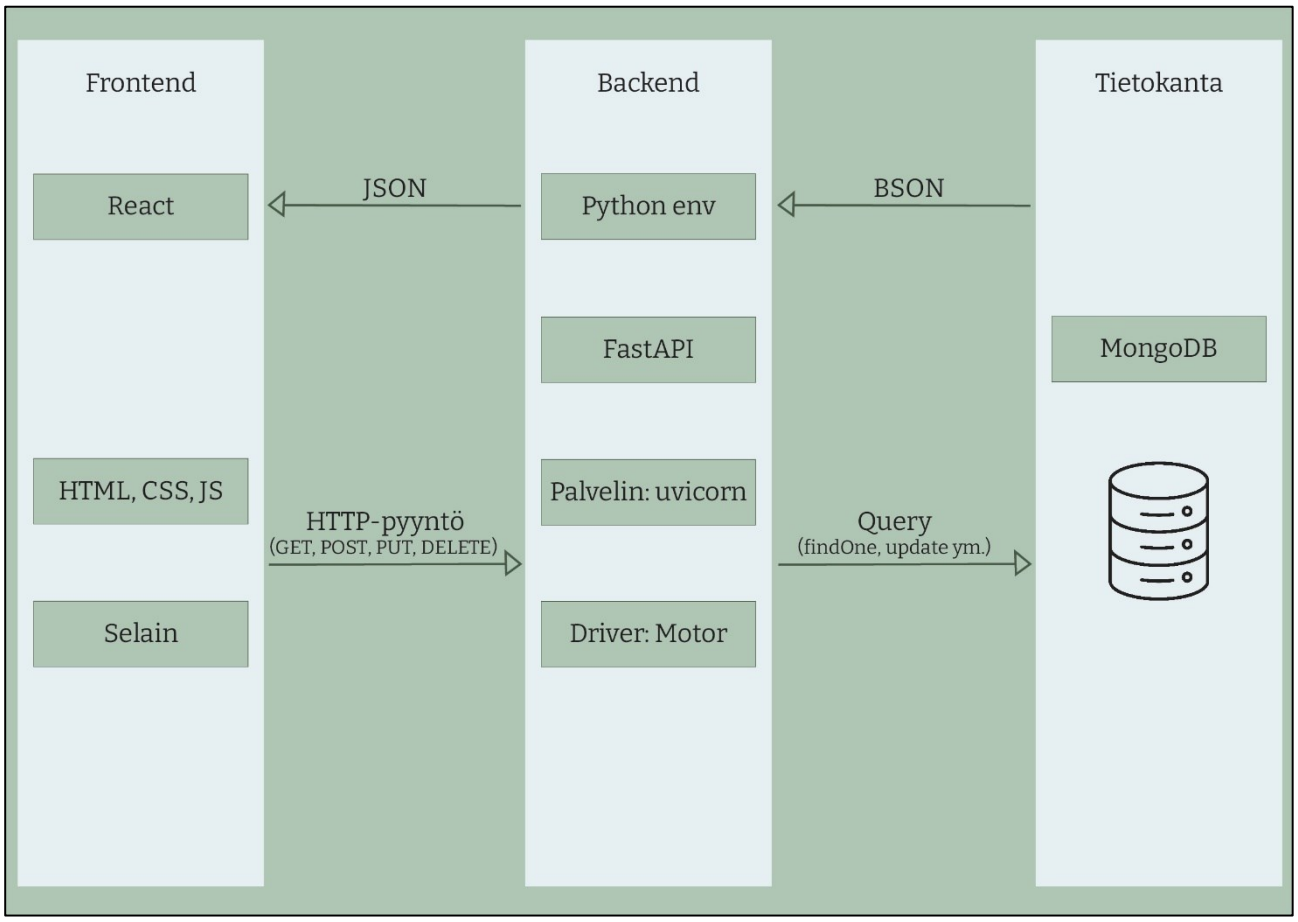
niputettua otsikkolistaksi ja toisella hyödynnettyä korttipohjaista rakennetta, jotta käyttäjä näkee yhdellä silmäyksellä tarjolla olevat aiheet/linkit.

Responsiivisuus suunniteltiin toteutettavaksi siten, että tablettikoossa navigaatiopalkin linkit kasaantuvat hampurilaisvalikkoon. Lisäksi kuvat skaalautuisivat SVG-muodossa suhteessa näyttökokoon. Flex-rakennetta hyödyntäen sivujen rakenne voisi muutamaa näyttökoon ns. breakpoint pistettä hyödyntäen kasaantua vertikaalisesti horisontaalisen levittäytymisen sijaan. Mobiilikokoisella näytöllä myös kuvat poistettaisiin, jotta tekstisisältö nousisi etusijalle.

Suunnitteluvaiheessa selvitettiin, millä keinoin voitaisiin päästä asetettujen tavoitteiden mukaisiin toimintoihin järkevimmin eli minkälainen ohjelmistopino (engl. stack) kannattaa valita. Valinnassa kiinnitettiin eniten huomiota omaan tekniseen osaamiseen, jotta projekti saadaan vietyä loppuun valituilla teknologioilla mahdollisimman tiukassa aikataulussa. Näin ollen priorisoitiin teknologioita, joista oli jo aiempaa kokemusta ja rajattiin ohjelmointikielet sellaisiin, joita olin jo aiemmin käyttänyt. Lisäksi verrattiin teknologioiden ominaisuuksia projektin tarpeisiin ja otettiin myös huomioon tavoite toteuttaa projekti ilmaiseksi tai mahdollisen vähäisin kustannuksin.

Suosittuja ohjelmistopinoja, joista lähdettiin hakemaan tälle projektille sopivaa, olivat mm. MERN (MongoDB. Express.js, React ja Node.js), MEAN (MongoDB, Express.js, Angular.js, Node.js), PERN (Postgres, Express.js, React, Node.js) ja LAMP (Linux, Apache, MySQL, PHP). Projektin kokonaisuus huomioon ottaen päädyttiin kokeilemaan toteutusta FARM pinolla (FastApi, React ja MongoDB).

FARM-pinon osaset voidaan ajatella kolmelle tasolle (kuva 2). Ensimmäisellä tasolla selainta käyttävä asiakas näkee Reactilla toteutetun visuaalisen käyttöliittymän, joka hyödyntää JavaScriptia, HTML:ää ja CSS:ää. Käyttäjän toimet laukaisevat HTTP-pyyntöjä REST API rajapinnan eli FastAPI:n käsittelyyn. Toisella tasolla sijaitseva FastAPI on käärittynä Python virtuaaliympäristöön ja tarvittaviin riippuvuuksiin. Tarvittavia Python paketteja ovat Uvicorn (ASGI-palvelin, mahdollistaa asynkronisen koodin käytön) ja Motor (asynkroninen Python-ajuri) muodostamaan yhteys tietokantaan kolmannella tasolla. MongoDB tietokantaan tulevien kyselyiden vastaukset ovat BSON-muodossa ja käsitellään lopulta FastAPI:ssa JSON muotoon. (Aleksendrić 2022)



Kuva 2. FARM pinon kolme tasoa

FastApi mahdollistaa nopean backend kehityksen python-ohjelmointikielellä, React helpottaa ja nopeuttaa frontend kehitystä ja yhteensopivuus näiden välillä tarjoaa helpon datan siirtymisen front- ja backendin välillä. MongoDB:n kohdalla mahdollisuus luoda tietokanta ilmaiseksi ja helposti pilvipalvelu MongoDB Atlasiin vaikutti suuresti sen valintaan. Tällä ohjelmistopinolla voitaisiin saavuttaa nopeutta kehittämistyöhön sekä suhteellisen helppo ylläpidettävyys (Aleksendrić 2022). Avoimen lähdekoodin ratkaisut mahdollistavat kustannusten pysymisen minimissä sekä sallivat lähdekoodin jakamisen alkuperäisenä ja muokattuna (Stallman 2021). Lisäksi näillä teknologioilla vaikutti olevan kattava dokumentaatio sekä paljon yhteisötukea (mm. StackOverflow kysymysten ratkaisut), joka voisi helpottaa kehitystyötä ongelmatilanteissa.

JavaScript-kirjastoa Reactia hyödyntäen voitaisiin toteuttaa haluttu navigaatio ilman sivujen uudelleen lataamista jokaisella linkin painamisella sekä kääriä koko sovellus haluttuun käyttöoikeussäännöt sisältävään kontekstiin. Itselleni oli tuttua ja luontevaa, että tietokantaa voisi käyttää API-rajapinnan avulla, joten FastApi vastaisi tähän tarpeeseen. Python-pohjainen FastAPI web-kehys tunnetaan erityisesti nopeudestaan ja suorituskyvystään (Aleksendrić 2022). NoSQL-tietokantana

MongoDB taipuisi mukavan joustavasti erilaisiin ja mahdollisesti muuttuviin sisältötarpeisiin. Se soveltuisi hyvin myös dokumenttipohjaisen tiedon tallentamiseen.

4.4 Ylläpito

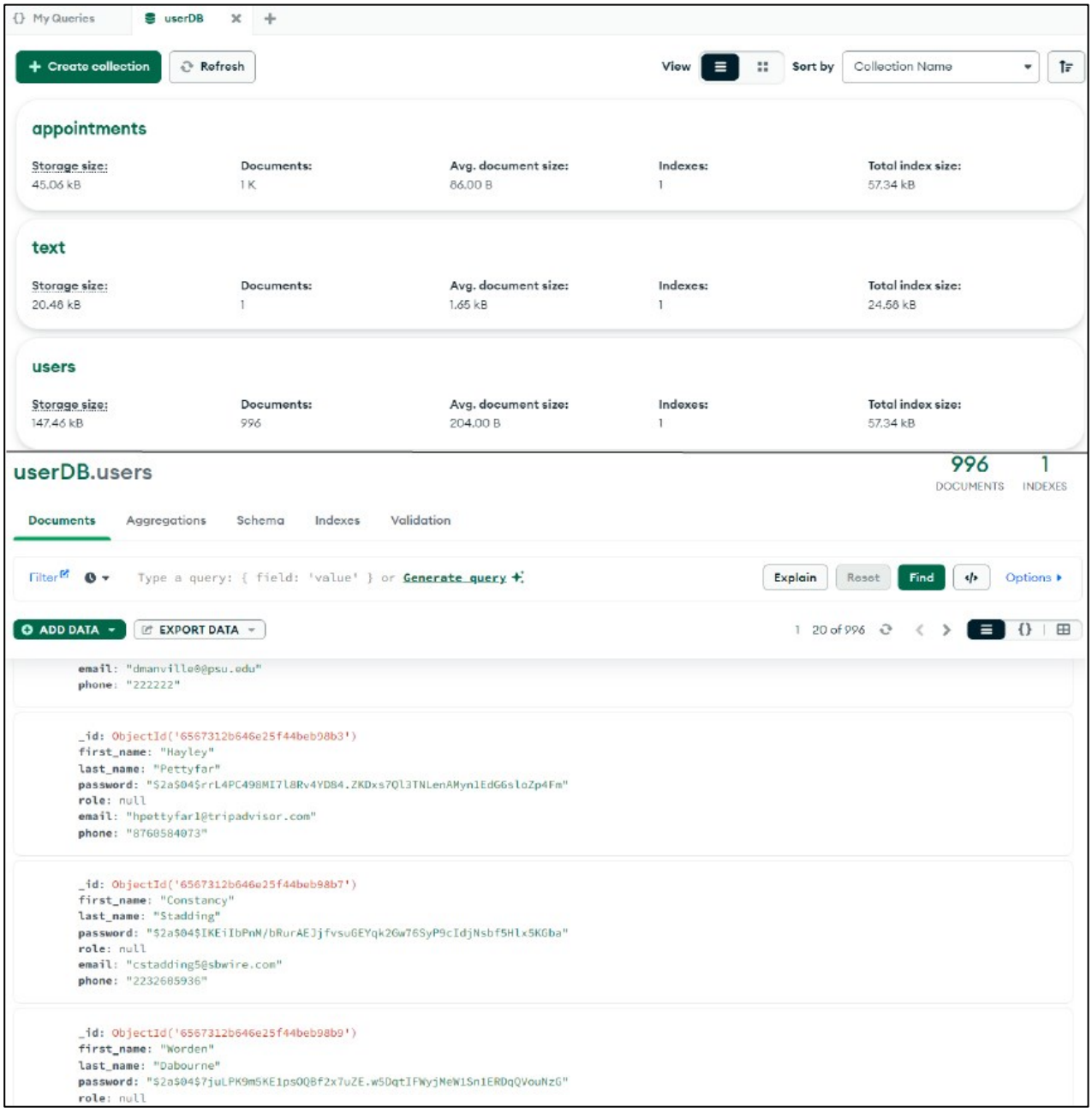
Jotta projektin lopputuloksena syntyvä verkkosivu vastaisi tarvetta olla psykoterapeutin työväline, tulisi miettiä myös ylläpidon toimimista valmiiden verkkosivujen julkaisun jälkeen. Potentiaalinen ammatinharjoittaja verkkosivujen omistajana ei todennäköisesti omaisi ohjelmointitaitoja tehdä muokkauksia sivuille, joten pyrittiin kartoittamaan mahdolliset päivitystarpeet. Päivitystarpeet liittyisivät todennäköisesti yhteystietojen muokkaamiseen, loma-aikojen tai muiden ajankohtaisten uutisten ilmoittamiseen ja toki käyttäjäasiakkaiden tietojen hallinnointiin.

Ylläpidon helpottamiseksi päädyttiin siihen, että tarvittavat tekstit sivuilla tallennettaisiin tietokantaan ja haettaisiin sieltä. Näin ollen teksti ei olisi staattisesti kiinni koodin seassa ja vaatisi koodin ymmärtämistä löytääkseen tekstit. Mikäli käyttäjä kävisi itse muokkaamassa toiminnallisuutta sisältävää kooditiedostoa, johon myös tekstit olisi kirjoitettu, olisi olemassa suuri virheiden riski. Tietokannasta haettavia tekstejä voitaisiin muokata tarpeen mukaan ja varmistettaisiin se, että sivuilla näkyisi aina viimeisin tieto. Verkkosivuille tulisi siis lisätä niin sanottu editointinäkymä tekstien muokkaamista varten. Selaimesta käytettävän muokkaustoiminnon avulla tapahtuva muutosten tekeminen olisi helppoa myös hatarat tietotekniset taidot omaavalle käyttäjälle.

5 Toiminnallinen osuus: Backend toteutus

5.1 MongoDB tietokannan perustaminen

Tietokantana projektissa käytettiin MongoDB tietokantaa Atlas pilvipalvelussa. Vaihtoehtona olisi ollut paikallisesti asennettu MongoDB, mutta pilvipalvelun käyttö vaikutti yksinkertaisimmalta vaihtoehdolta ensi kertaa käytettäessä ja ohjeet käyttöönottoon olivat kattavat. Rekisteröityessä MongoDB Atlas käyttäjäksi on mahdollista valita ilmainen ”Shared Cluster” ja siihen ”M0 Sandbox” vaihtoehto. (MongoDB s.a.)



Kuva 3. Compass, yllä näkymä datakokoelmista ja alla käyttäjäkokoelman harjoitusdatasta

Tietokannan hallinnointiin Windowsilla käytettiin Compass-työpöytäsovellusta, joka muodostaa yhteyden pilvipalvelussa sijaitsevaan tietokantaan Atlas-palvelussa luodulla käyttäjätunnuksella ja salasanalla. Compass-ohjelmaa käyttäen luotiin tietokanta ja luotuun tietokantaan lisättiin kolme tietokokoelmaa: käyttäjät (users), ajanvaraukset (appointments) ja tekstit (texts). MongoDB Compass-ohjelmassa on mahdollista muokata dataa helposti ja tehdä erilaisia hakuja/kyselyitä.

Tietokokoelmiin on mahdollista lisätä dataa tallentamalla sinne JSON-muotoisia dokumentteja, joten käyttäjistä ja ajanvarauksista lisättiin harjoitusdataa, jotka luotiin Mockaroo internetsivuilla (Mockaroo s.a.) tiettyjen valittujen kenttien mukaan. Näin tietokantatoimintojen testaaminen oli helpompaa alkuvaiheessa.

Käyttäjätietoihin jokaisesta käyttäjästä tallennettiin avain-arvopareina etunimi (”first_name”), sukunimi (”last_name”), puhelinnumero (”phone”), sähköposti (”email”), salasana (”password”) ja rooli (”role”) -kentät. Ajanvarauksista puolestaan päivämäärä (”date”), kellonaika (”time”), kesto (”duration”) ja totuusarvomuuttuja ”true”/”false” sen mukaan, onko aika peruutettu vai ei (”cancelled”). Teksteistä tallennettiin kentät otsikko (”header”) ja sisältö (”body”). Jokaiselle dokumentille luodaan automaattisesti yksilöllinen id-tunnus, sitä ei ollut tarpeen määritellä itse dataa luotaessa.

5.2 FastApin päätepisteet

Tietokannan perustamisen jälkeen voitiin aloittaa itse backendin rakentaminen. Ohjelmointiin käytettiin ilmaista Visual Studio Code-editoria ja versionhallintaan GitHubia. VS Code sisältää Git-integroinnin ja GitHub-sisäänkirjautumisen jälkeen voi päivitettyjä koodiversioita siirtää omaan Github repositorioon suoraan terminaalikomennoilla.

FastApin käyttöön ottaminen aloitettiin luomalla projektin työskentelykansion sisälle Python virtuaaliympäristö venv-ohjelmistolla ja aktivoimalla luotu virtuaaliympäristö. Tähän virtuaaliympäristöön asennettiin PIP-työkalua käyttäen tarvittavat Python-paketit ja riippuvuudet, kuten FastApi, Python-kirjastot Uvicorn, Motor ja myöhemmin myös Pydantic.

Asennettu FastApi-luokka tuotiin Python tiedostoon ”main.py” ja luotiin app-muuttujana, jonka kautta toiminnallisuutta voitiin käyttää. FastApin yhdistäminen MongoDB tietokantaan tehtiin Motorin avulla (Motor 3.3.2 documentation s.a.) ja tietokannan määrittelyt asetettiin ympäristömuuttujaan.

```
from decouple import config
from motor.motor_asyncio import AsyncIOMotorClient

DB_URL = config('DB_URL', cast=str)
DB_NAME = config('DB_NAME', cast=str)

@asynccontextmanager
async def lifespan(app: FastAPI):
    app.mongodb_client = AsyncIOMotorClient(DB_URL)
    app.mongodb = app.mongodb_client[DB_NAME]
    yield
    app.mongodb_client.close()
```

Kuva 4. Tietokantaan yhdistäminen main.py tiedostossa

CORS (Cross-Origin Resource Sharing) säätelee, miten verkkosivustot voivat pyytää resursseja toisilta verkkotunnuksilta. Koska tämän projektin frontend ja backend toimivat eri verkkotunnuksilla, tuntui helpommalta sallia kehitysvaiheessa kaikki yhteydet, jotta virheet CORS-määrittelyissä eivät hankaloittaisi toiminnallisuuksien testaamista. Myöhemmässä vaiheessa CORS-määrittelyt voitaisiin tiukentaa ja rajoittaa sallittuja alkuperiä.

```
app = FastAPI(lifespan=lifespan)

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Kuva 5. FastAPI:n luominen, CORS määrittelyssä sallittiin kehitysvaiheessa kaikki yhteydet

Projektissa käytettävät API päätepisteet olivat käyttäjiin liittyvät ”/user” kaikkien käyttäjien listaamiseen (get-metodi), ”/user/login” sisäänkirjautumiseen (post-metodi), ”/user/register” rekisteröintiin ja ”/user/me” kirjautuneen käyttäjän autentikointiin ja käyttäjätietojen hakemiseen (get-metodi). Lisäksi käyttäjän päivittämiseksi ja poistamiseksi luotiin "/user/{id}" päätepisteeseen put ja delete -metodit.

```python
@router.get("/", response_description="List all users")
async def list_all_users(request: Request) -> List[UserDB]:
    result = request.app.mongodb['users'].find({})
    results = [UserDB(**raw_user) async for raw_user in result]
    return results
```

Kuva 6. Kaikki käyttäjät hakeva get-päätepiste

Ajanvaraustietojen hakemiseen tehtiin päätepisteeseen ”/appt” get ja post -metodit kaikkien varattujen aikojen hakemiseen ja uuden ajan luomiseen. Post-metodi vaati käyttäjän olevan kirjautuneena sisään, jotta autentikoinnin tuloksena saatu id-tunnus lisättiin vierasavaimeksi ajanvarauksen ”customer”-kentäksi. Lisäksi luotiin "/appt/{id}" päätepiste yksittäisen asiakkaan ajanvarausten hakemiseen admin-oikeuksilla toimivalle käyttäjälle.

Verkkosivun tekstien tietokantaan lisäämiseksi, tallentamiseksi ja muokkaamiseksi tehtiin ”/text”-päätepisteeseen get ja post-metodit kaikkien tietokantaan tallennettujen tekstien listaamiseksi ja uuden tekstin tallentamiseksi. Lisäksi "/text/{page}" päätepisteeseen luotiin get-metodi tekstien hakemiseksi sivukohtaisesti. ”/text/{id}” päätepisteen put-metodilla yksittäisiä tekstejä pystyy muokkaamaan halutunlaiseksi.

Pydanticia hyödynnettiin tiedon validointiin ja mallien määrittelyyn tietokantakokoelmien käyttöä varten. Esimerkiksi käyttäjätietojen kenttien sisältö määriteltiin alla olevan kuvan 7 mukaisesti helpottamaan tietokantataulujen käyttämistä tietoa etsittäessä, tallennettaessa ja palautettaessa. Virtuaaliympäristössä käynnistetty Uvicorn seurasi ja ajoi muutoksia python-koodissa. Jotta luotuja päätepisteitä voitiin testata, käytettiin Postmania REST clientina.

```python
from bson import ObjectId
from enum import Enum
from pydantic import EmailStr, Field, BaseModel, ValidationError

class Role(str, Enum):
    USER = "user"
    ADMIN = "admin"

class UserBase(BaseModel):
    first_name: str = Field(..., description="The first name of the user")
    last_name: str = Field(..., description="The last name of the user")
    password: str = Field(..., description="The user's password")
    role: Role = Role.USER
    email: EmailStr = Field(..., description="The email address of the user")
    phone: str = Field(..., description="The phone number of the user")
```

Kuva 7. Käyttäjätietojen mallin määrittely

Yllä kuvailtujen päätepisteiden metodeissa tallennetaan myös käyttäjän syöttämiä tietoja tietokantaan. Jotta käyttäjä ei pysty vahingossa tai tarkoituksellisesti tallentamaan tietokantaan haitallista HTML-koodia (XSS-hyökkäys, engl. Cross-Site Scripting), lisättiin koodiin HTML-tagit syötteestä puhdistava toiminto. Tämä toteutettiin yksinkertaisesti poistamalla halutut merkit käyttäjän syötteestä ”replace”-metodilla.

```
@router.post("/", response_description="Add new text")
async def create_text(request: Request,text: TextBase = Body(...)):
    text = jsonable_encoder(text)

    # Clean up HTML tags in all fields
    for key, value in text.items():
        if isinstance(value, str):
            text[key] = value.replace("<[^>]*>", "")

    new_text = await request.app.mongodb["text"].insert_one(text)
    created_text = await request.app.mongodb["text"].find_one({"_id": new_text.inserted_id})
    return JSONResponse( status_code=status.HTTP_201_CREATED,
        content=created_text)

@router.put("/{id}", response_description="Update text")
async def update_text(request: Request, id: str, text: TextUpdate = Body
         (...)):
    text_update_dict = text.dict(exclude_unset=True)

    # Clean up HTML tags in all fields
    for key, value in text_update_dict.items():
        if isinstance(value, str):
            text_update_dict[key] = value.replace("<[^>]*>", "")

    if text_update_dict:
        update_result = await request.app.mongodb["text"].update_one({"_id": id}, {"$set": text_update_dict})

        if update_result.modified_count == 1:
            return

    raise HTTPException(status_code=404, detail=f"text {id} not found")
```

Kuva 8. HTML-tagien puhdistaminen käyttäjän syötteestä verkkosivun tekstejä luotaessa ja muokattaessa (merkitty kommenteilla) ennen tallentamista tietokantaan.

5.3 Käyttäjävarmennus

Käyttäjävarmennus toteutettiin vertaamalla käyttäjän kirjoittamaa sähköpostia ja salasanaa tietokantaan tallennettuihin. Käyttäjää rekisteröitäessä salasana tallennetaan Passlib-kirjaston hajautusalgoritmeja (engl. hashing algorithm) hyödyntäen kryptattuna tietokantaan ja kirjautumisen yhteydessä verrataan täsmäävätkö salasanat kyseisen sähköpostin kanssa tallennettuun (PassLib s.a.).

```
import jwt
from fastapi import HTTPException, Security
from fastapi.security import HTTPAuthorizationCredentials, HTTPBearer
from passlib.context import CryptContext
from datetime import datetime, timedelta
from decouple import config

class AuthHandler:

    security = HTTPBearer()
    pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
    secret = config('SECRET', cast=str)

    def get_password_hash(self, password):
        return self.pwd_context.hash(password)

    def verify_password(self, plain_password, hashed_password):
        return self.pwd_context.verify(plain_password, hashed_password)

    def encode_token(self, user_id):
        payload = {
            "exp": datetime.utcnow() + timedelta(days=0, minutes=30),
            "iat": datetime.utcnow(),
            "sub": user_id,
        }
        return jwt.encode(payload, self.secret, algorithm="HS256")
```

Kuva 9. JWT-avaimen muodostaminen

Todennetulle käyttäjälle asetetaan aikarajattu jwt-avain (JSON Web Token) PyJWT-kirjastoa käyttäen (PyJWT s.a.). Jwt-avain koostuu pisteillä erotettuna otsakkeesta (Javascript Object Signing and Encryption -otsake), avaimen tiedoista (payload) ja allekirjoituksesta (Hakatemia 2022). Otsakkeessa on tieto varmennusavaimen salausalgoritmista.

Yleisimmät käytetyt algoritmit ovat epäsymmetrinen RS256 (RSA Signature with SHA-256) ja symmetrinen HS256 (HMAC with SHA-256). Nyt päädyttiin käyttämään jälkimmäistä, joka käyttää samaa algoritmia luomaan ja varmistamaan varmennusavaimen allekirjoituksen. Näin tehdessäni itse

sekä front- että backendin tarvitsi hallita vain yhtä avainta kahden sijaan, sillä mukana ei ole muita osapuolia, joille avaintenhallintaa tarvitsisi jakaa. (Chim 24.9.2020)

Varmennusavaimen tietoihin tallennettiin käyttäjän id, aikaleima avaimen myöntämishetkestä ja varmennusavaimen vanhentumisaika. Allekirjoitus muodostuu otsakkeen, tietojen ja erillisen salausavaimen yhdistelmästä, joka on ajettu valitun salausalgoritmin (HS256) läpi (Hakatemia 2022). Yllä olevasta kuvasta 9 näkyy jwt-avaimen muodostaminen, jossa salausavain (muuttuja nimeltä ”secret”) on määritelty piiloon ympäristömuuttujiin.

Käyttäjäntodennusta käytettiin tiettyjen päätepisteiden rajaamiseen pois kirjautumattomilta tai väärän roolin omaavilta käyttäjiltä. Tämä toteutettiin varmennusmuuttujan käyttämistä riippuvuutena päätepistettä määriteltäessä. Kuvassa 10 näkyvässä päätepisteessä käyttäjän id:n hakeminen varmennusavaimen tiedoista riippuu siitä, onko PyJWT:n avulla purettu varmennusavain oikea ja voimassa.

```
# users.py

    @router.get("/me", response_description="Logged in user data")
    async def me(request: Request, userId=Depends(auth_handler.auth_wrapper)):
        currentUser = await request.app.mongodb["users"].find_one({"_id": userId})
        result = CurrentUser(**currentUser).model_dump()
        result["id"] = userId
        return JSONResponse(status_code=status.HTTP_200_OK,
            content=result)

    # authentication.py

    def decode_token(self, token):
        try:
            payload = jwt.decode(token, self.secret, algorithms=["HS256"])
            return payload["sub"]
        except jwt.ExpiredSignatureError:
            raise HTTPException(status_code=401, detail="Signature has expired")
        except jwt.InvalidTokenError as e:
            raise HTTPException(status_code=401, detail="Invalid token")

    def auth_wrapper(self, auth: HTTPAuthorizationCredentials = Security(security)):
        return self.decode_token(auth.credentials)
```

Kuva 10. Käyttäjätodennus users.py ja authentication.py -tiedostoissa

5.4 Sovelluksen siirto Herokuun

Projektin API backend ladattiin Heroku-pilvipalveluun käyttäen Git:iä ja Heroku CLI työkalua, jonka avulla sovelluksia voi luoda ja hallinnoida suoraan terminaalista (Heroku Dev Center 2023a). Heroku on pilvipalvelu, joka tarjoaa yksinkertaisen ja edullisen tavan julkaista sovelluksia. Tässä projektissa Herokua käytettiin backendin julkaisuun, mutta Herokuun voisi halutessaan julkaista myös koko sovelluksen.

Kuva 11. Procfilen sisältö

Palvelimelle siirron onnistuminen vaati Procfile ja requirements.txt -tiedostojen luomista. Requirements.txt listaa käytetyt kirjastot eli tarvittavat riippuvuudet sovelluksen toimimiseksi. Projektin juureen sijoitettava tekstipohjainen Procfile-tiedosto kertoo, miten sovellus tulisi käynnistää ja mitä prosesseja tarvitaan sovelluksen toimimiseksi. Nyt luotu kuvan 11 mukainen Procfile määrittää sanalla ”web” käytetyn dynon käsittelevän HTTP-pyyntöjä. Loppuosa on käynnistyskomento, joka käynnistää Uvicornin, ottaa käyttöön ”main”-moduulin ja sen ”app”-objektin ja varmistaa, että sovellus on saatavilla kaikille verkossa (”--host=0.0.0.0”). Koska Herokun sovelluksen portti voi vaihdella, kuunneltava portti on määritelty dynaamisesti ympäristömuuttujan avulla (”$PORT”).

Versionhallinnan ollessa ajan tasalla Githubissa backend voitiin ladata Herokuun kirjautumalla sisään kuoressa (engl. shell) ja luomalla Heroku sovellus. Samalla Herokuun luodaan etärepositorio, johon tulevat muutokset voidaan ajaa samoin kuin aiemmin Githubiin. Salaiset ympäristömuuttujat tuli lisätä Herokuun vielä erikseen. Lopulta API toimi Herokun palvelimella omassa osoitteessaan main-tiedostossa määriteltyjen päätepisteiden kanssa. (Heroku Dev Center 2023b)

Palvelimelle siirron onnistuttua backend saatiin käynnistettyä ”heroku open” komennolla ja sen toimivuutta testattiin Postmanilla. APIn toimintaa ja virhetilanteita seurattiin ”heroku logs –tail” komennolla ja tarvittavien muutosten jälkeen tehtiin aina uusi add, commit ja push sekä GitHub että Herokun omiin etärepositorioihin, jonka jälkeen sovellus käynnistettiin uudelleen.

6 Toiminnallinen osuus: Frontend toteutus

Verkkosivujen backendin ollessa pääosin valmis, aloitettiin frontend osuus, joka toteutettiin käyttäen React.js kirjastoa. React ja muut käytetyt kirjastot asennettiin npm-paketinhallintatyökalulla. Muotoilussa hyödynnettiin React Bootstrapia, joka tarjoaa paljon valmiita komponentteja käyttöliittymän tyylittelyyn.

6.1 Navigaatio

Navigaatiopalkin linkkien polut määriteltiin React Router -kirjaston avulla. Toteutettaessa navigaation reititys näin, sivusto ladataan vain yhden kerran ja linkkien klikkaaminen käynnistää vain tarvittavien komponenttien päivittämisen JavaScriptin avulla. Routerin etuja ovat mm. dynaamisten reittien mahdollistaminen, parametrien välittämisen mahdollistaminen helposti ja datan lataamisen integroiminen reitteihin (Roldán 2023, luku 9). Reititys toteutettiin kuvan 12 mukaisesti BrowserRouter komponentilla, jonka sisälle määriteltiin Route-komponentit. Eri roolissa selaavien käyttäjien erilainen reititys määriteltiin käärimällä ne autentikointikomponentin sisään. Tämä komponentti tarkistaa käyttäjän oikeudet ja ohjaa oikealle sivulle.

```
<BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="page1" element={<Page1 />} />
          <Route path="page2" element={<Page2 />} />
          <Route path="login" element={<Login />} />
          <Route path="register" element={<Register />} />
          <Route path="*" element={<NoPage />} />

          <Route element={<RequireAuthentication />}>
            <Route path="user" element={<User />} />
            <Route path="appt" element={<UserAppt />} />
            <Route path="createappt" element={<UserCreateAppt />} />
            <Route path="admin" element={<Admin />} />
            <Route path="customers" element={<AdminCustomers />} />
            <Route path="appointments" element={<AdminAppointments />} />
            <Route path="createappointments" element={<AdminCreateAppointments />} />
          </Route>
        </Route>
      </Routes>
    </BrowserRouter>
```

Kuva 12. Reititys React Router-kirjaston avulla

Vierailevan selaajan näkökulmasta reittejä verkkosivuilla on neljä/viisi: etusivu, kaksi vaihtoehtoista pohjaa tiedon jakamiselle ja kirjautumissivu/rekisteröitymissivu. Näistä kirjautumisvalikko

toteutettiin React Icons -kirjaston lukkoa esittävän ikonin alle pudotusvalikkona. Kirjautuneelle käyttäjälle vastaava kohta esittää auennutta lukkoa, jonka alta löytyvät linkit omiin tietoihin, ajanvarauksiin sekä uuden ajan varaamiseen pudotusvalikkona.

Projektin alussa GitHub Pages palvelua suunniteltiin käytettävän verkkosivun staattisen frontendin hostaamiseen. Perusteena Pagesin valinnalle oli helppo ja intuitiivinen yhteensopivuus GitHubin kanssa, mihin verkkosivujen kaikki koodi oli ladattu Git-versionhallinnan tarkoituksessa (Uotila 2021). Useiden kokeilujen ja virheilmoitusten selvittelyjen jälkeen kuitenkin selvisi, että GitHub Pages ei tue BrowserRouterin käyttöä. Myöskään vastaavalla tavalla toimivaa HashRouteria ei saatu toimimaan yrityksistä huolimatta. Koska Reactille sopivan reitityksen käyttöä pidettiin tärkeänä osana verkkosivujen toiminnallisuutta, päädyttiin vaihtamaan host alusta Azureen.

6.2 Jatkuva siirto Azureen

Paikallisesti ”localhost” palvelimella toimiva verkkosivu ei vielä riitä varmentamaan koodin toimivuutta oikeissa käyttöolosuhteissa. Siitä syystä sovellus olisi hyvä asettaa palvelimelle ja varmistaa, että se rakentuu oikein tuotantoon. Jokainen muutos on kuitenkin riski virheille koodissa ja mitä enemmän muutoksia tulee tehtyä ennen niiden tuotantoon laittamista, sitä vaikeampi on jäljittää niitä korjaamista varten. Jatkuva (pilvi)palvelimelle siirto on keino pienentää eroa paikallisen version ja tuotannossa olevan version välillä. (Nygard 2018, luku 13)

Tässä projektissa käytettiin Microsoft Azuren palveluita ja pilvessä ajettavaan palvelimeen siirtäminen tapahtui suoraan Githubiin ladattujen versioiden perusteella. Azure tarjoaa joitain ilmaisia palveluita, mutta lisäksi opiskelijana on mahdollista lunastaa krediittejä palvelujen käyttämistä varten. Tunnusten luomisen jälkeen Azureen voi luoda staattisen web-sovelluksen lisäämällä tiedot halutusta Github repositoriosta.

Azure luo sovelluksen tietyn työnkulun (engl. workflow) perusteella ja tätä työnkulku-tiedostoa voi muokata tarpeen mukaan (kuva 13). Tärkeää on kiinnittää huomiota mm. oikeisiin tiedostopolkuihin sekä työnkulun etenemisjärjestykseen. Virheilmoitukset osoittavat kohdan, jossa työnkulku on keskeytynyt ja mahdollistavat korjaustoimenpiteiden kohdistamisen oikeaan asiaan.

```yaml
name: Azure Static Web Apps CI/CD

on:
  push:
    branches:
      - master
  pull_request:
    types: [opened, synchronize, reopened, closed]
    branches:
      - master

jobs:
  build_and_deploy_job:
    if: github.event_name == 'push' || (github.event_name == 'pull_request' && github.event.action != 'closed')
    runs-on: ubuntu-latest
    name: Build and Deploy Job
    steps:
      - uses: actions/checkout@v3
        with:
          submodules: true

      - name: Build And Deploy
        id: builddeploy
        uses: Azure/static-web-apps-deploy@v1
        with:
          azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN }}
          repo_token: ${{ secrets.GITHUB_TOKEN }}
          action: "upload"
          app_location: "/" # App source code path
          api_location: "" # Api source code path - optional
          output_location: "build" # Built app content directory - optional

  close_pull_request_job:
    if: github.event_name == 'pull_request' && github.event.action == 'closed'
    runs-on: ubuntu-latest
    name: Close Pull Request Job
    steps:
      - name: Close Pull Request
        id: closepullrequest
        uses: Azure/static-web-apps-deploy@v1
        with:
          app_location: "/" # App source code path
          azure_static_web_apps_api_token: ${{ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN }}
          action: "close"
```

Kuva 13. Azure workflow:n määrittävä tiedosto, josta on poistettu kirjautumistiedot

Ensimmäisen onnistuneen siirron jälkeen paikallista version työstämistä voi jatkaa luottavaisin mielin, kunhan muistaa tallentaa ja julkaista tehdyt muutokset riittävän usein. Jokainen puskettu julkaisu Github etärepositorioon käynnistää työnkulun ja palvelimelle siirron uudestaan ja sovelluksen tuotantoversio päivittyy jatkuvasti.

6.3 Käyttäjävarmennus

Käyttäjävarmennus toteutettiin luomalla ensimmäiseksi kirjautumissivu, missä kävijä täyttää lomakkeeseen sähköpostinsa ja salasanan. Lomakkeen lähettämispainike käynnistää varmennuksen API-kutsulla POST-metodia käyttäen lähettäen sähköpostin ja salasanan verrattavaksi ”users/login” päätepisteeseen.

Mikäli pyyntö ja varmennus onnistuvat, pyyntö palauttaa varmennusavaimen (engl. access token). Tämä avain asetetaan evästeeksi (eng. cookie) ja sen lisäksi haetaan käyttäjäpäätepisteestä ”/me” käyttäjän nimitiedot ja asetetaan niiden lisäksi tieto sisäänkirjautumisesta autentikointikontekstiin, jota käytetään mukautetun React-koukun avulla (custom hooks) (Roldán 2023, luku 8). Tämän autentikointikontekstin päivittyminen käynnistää myös navigaatiopalkin sisällön päivittymisen ja vaihtaa sisään kirjatulle käyttäjälle sopivat kuvakkeet ja linkit näkyville.

Vierailijalle näkyvät sisällöt ja linkit ovat tietosisällöt tarjotuista palveluista ja ammatinharjoittajasta. Sen lisäksi, että navigaatiopalkissa näkyvät eri linkit kirjautumistilanteen mukaan, kirjautumisen alle suojatut reitit ovat käärityt kirjautumisstatuksen tarkastavaan tiedostoon (kuva 14). Tämä tiedosto tarkistaa, löytyykö käyttäjältä voimassa oleva varmennusavain evästeistä ja ohjaa oikealle sivulle sen löytyessä. Mikäli vierailija kokeilee suoraan selaimen osoiteriville jotain kirjautuneelle käyttäjälle suunnattua päätepistettä, hänet ohjataan kirjautumissivulle. Tämä toteutettiin kietomalla suojatun navigaation osa käyttäjävarmennuksen tekevään elementtiin, joka tarkistaa voimassa olevan varmennusavaimen evästeestä sekä kirjautuneen käyttäjän roolin.

```
import { Navigate, Outlet } from "react-router-dom";
import useAuth from "../hooks/useAuth";

const RequireAuthentication = () => {
  const { isLoggedIn, auth } = useAuth();

  return isLoggedIn && auth.role=="admin" ? <Outlet /> : <Navigate to="/login" />;
};

export default RequireAuthentication;
```

Kuva 14. RequireAuthentication.js tiedosto tarkistaa useAuth-koukun avulla, onko käyttäjä kirjautunut admin-oikeuksin ja ohjaa sen mukaan oikealle sivulle

Kirjautuneelle käyttäjälle, jonka rooliksi on asetettu käyttäjä (”user”), näkyvät sen lisäksi hänen omat käyttäjätietonsa ja hänelle varatut ajat. Käyttäjä voi varata itselleen uusia aikoja, jotka tallentuvat tietokantaan tallentaen vierasavaimena käyttäjän id:n numero-kirjainsarjan. Käyttäjältä on kuitenkin rajattu pois mahdollisuus muokata ajanvarauksen tietoja.

Admin-roolissa olevalle kirjautuneelle käyttäjälle tulee lisäksi näkyviin uusi navigaatiovalikko ja hän pystyy hallinnoimaan käyttäjien tietoja, ajanvarauksia sekä verkkosivujen tekstejä. Käyttäjien tiedot asetettiin taulukkoon ”/users”-päätepisteen GET-haun jäljiltä ja lisäksi jokaiselle tietoriville lisättiin React Icons -kirjastosta poisto- ja muokkaustoimintoja kuvaavat ikonit, joita klikkaamalla voi poistaa tai muokata kyseisen käyttäjän tietoja. Poisto-toiminto käyttäjän tulee varmentaa pop-up ikkunan vahvistuspainikkeella ja Muokkaus-toiminto avaa tiedot niin-ikään pop-up ikkunaan, jossa tiedot ovat muokattavassa lomakenäkymässä. Vahvistus-painike lähettää muokatut tiedot PUT-metodilla API:lle ja päivittää taulukon tiedot.

6.4 Visuaalinen toteutus

Käyttöliittymän ulkoasun värit ja tyyli määriteltiin aluksi tehdyn suunnitelman mukaan CSS-tiedoston määritysten sekä Bootstrap luokkanimien avulla. Web-sovelluksen sisältö käärittiin erillisten Layout, Header ja Footer-tiedostojen sisälle, jotta muotoiluja ei tarvinnut toistaa montaa kertaa. CSS-tiedostoon määriteltiin eri html-elementeille värimaailma ja tyyli sekä käytetyt Adobe-fontit. Bootstrapista hyödynnettiin valmiita toiminnallisuuksia, kuten container-säiliöiden responsiivista grid-asettelua, navigaatiokomponenttia ja alaspudotusvalikoiden tyylittelyjä.

Aloitussivulle aseteltiin kolme call-to-action -painiketta, jotka ohjaavat käyttäjän nopeasti toivotuille sivuille. Vierailijalle suunniteltiin näkyviin selkeä muutaman linkin navigaatio sekä lukkoikonin alle kirjautumismahdollisuus. Kirjautumisstatuksen perusteella admin-oikeuksin kirjautuneelle tulee näkyviin myös lisälinkit ”asiakkaat”, ”ajanvaraus” ja ”tee uusi ajanvaraus” navigaatioon. Nämä sisältävät asiakas- ja ajanvarauslistat sekä lomakkeen uusien ajanvarausten tekoon. Lomakkeella on pudotusvalikko, johon on tietokannasta haettu kaikki asiakkaat ja sieltä admin voi valita asiakkaan, jolle ajanvaraus tehdään.

Kuva 15. Aloitussivu, jossa hiiri on kolmannen CTA-painikkeen päällä. Työotsikkona toimivan Terapiatertun sivuille on kirjauduttu admin-tunnuksin.

Kuva 16. Tietosivu 1. Sivuilla vierailee kirjautunut käyttäjä, joka on klikannut pudotusvalikon auki oikealla yläreunassa.

Ensimmäiselle tietoa sisältävälle sivuille aseteltiin sekä ”laatikkovalikko”, josta otsikoiden mukaista tietosisältöä voi klikata esille, että korttikomponentti kuvalla ja tekstillä. Toiselle tietosivulle tekstisisältöä jaoteltiin viiteen sarakkeeseen ja kuva oikeaan yläreunaan. Adminilla on näkyvissä myös muokkausmahdollisuus, joka aukeaa pop-up-lomakkeena kunkin tekstin alla olevasta painikkeesta (kuva 17). Tekstikappaleita on mahdollista myös poistaa ja lisätä lomakkeen avulla.

Kuva 17. Ylläpidon helpottamiseksi tekstejä on mahdollista muokata painikkeesta aukeavan lomakkeen avulla

7 Pohdinta

Tämän opinnäytetyön tavoitteena oli suunnitella ja toteuttaa verkkosivut psykoterapian työn tueksi markkinoimaan tarjottuja palveluita ja tarjoamalla asiakasaikojen hallinnointiin työkaluja. Työ aloitettiin perehtymällä teoriaan verkkosivujen suunnittelusta ja toteutuksesta. Tämän jälkeen määriteltiin projektin tavoitteet ja lähtökohdat sekä kohdeasiakkaan että terapeutin näkökulmasta, jotta voitiin suunnitella verkkosivut. Toteutusvaiheessa työskentelyssä seurattiin laadittua suunnitelmaa, mutta tehtiin myös käytännön myötä huomattujen haasteiden vuoksi muutoksia alkuperäiseen suunnitelmaan.

Opinnäytetyön sisältö on ajankohtainen, sillä terveydenhuollon digiloikan myötä myös yksityisten terapeuttipalveluja tarjoavien tulee mahdollistaa palvelujen saatavuus verkossa sekä panostaa digitaalisiin ratkaisuihin. Palvelujen markkinoimisen ja tarjoamisen ensimmäinen askel ovat toimivat verkkosivut, joihin voidaan jatkossa integroida esimerkiksi etätapaamisten mahdollisuus.

Projektin lopputuloksena syntyivät verkkosivut, joiden toiminnallisuus ja ulkonäkö vastasi odotuksia ja asetettuja tavoitteita. Syntyneiden verkkosivujen koodipohjaa on mahdollista hyödyntää pienin muutoksin tulevaisuudessa joko psykoterapeutin verkkosivuiksi tai muun pienyrittäjän verkkosivuina. Sivuille on mahdollista lisätä nykyistä kattavammat ajanvarauspalvelut tai esimerkiksi asiakkaalle mahdollisuus täyttää ja tallentaa erilaisia lomakekyselyitä, jota käytetään terapiatyöskentelyn tukena. Nämä lisäykset vahvistaisivat entisestään verkkosivujen hyödyllisyyttä psykoterapeutin työn tukena.

Opinnäytetyön kokonaisuutta pala palalta työstäessäni jouduin monesti oman mukavuusalueeni ulkopuolelle uuden edessä ja opin paljon. Käytännön teknisiä taitoja opin tekemisen kautta ja ongelmatilanteissa tietoa etsiessäni. Sain kokeilla itsenäisesti opinnoissa hankkimaani osaamista sekä syventää sitä käyttämieni teknologioiden osalta. Projektin myötä hahmotin selkeämmin omat ammatilliset kiinnostuksen kohteeni ja uskon, että valmistumisen jälkeen on nyt helpompi hakeutua itseä kiinnostavalle työpolulle.

Lähteet

Aleksendrić, M. 2022. Full Stack FastAPI, React, and MongoDB: Build Python Web Applications with the FARM Stack. Packt Publishing. E-Kirja. Luettu: 19.1.2024.

Aluehallintovirasto s. a. Saavutettavuus. Luettavissa: https://avi.fi/tietoa-meista/tehtavamme/saavutettavuus. Luettu: 19.1.2024.

Aluehallintovirasto s. a. Ohjeita suunnittelun tueksi. Luettavissa: https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/ohjeita-suunnittelun-tueksi/. Luettu: 19.1.2024.

Aluehallintovirasto s. a. Yleistä saavutettavuudesta. Luettavissa: https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/. Luettu: 19.1.2024.

Ardeleanu, S. 2016. Relational Database Programming: A Set-Oriented Approach. Apress. E-kirja. Luettu: 19.1.2024

Axios s. a. Documentation: Getting started. Luettavissa: https://axios-http.com/docs/intro. Luettu: 15.1.2024.

Beaird, J., Walker, A. & George, J. 2020. The Principles of Beautiful Web Design. 4. painos. SitePoint Pty. Ltd. Australia. E-Kirja. Luettu: 15.1.2024.

Bierer, D. 2020. Learn MongoDB 4.x: A guide to understanding MongoDB development and administration for NoSQL developers. Packt Publishing, Limited. E-kirja. Luettu: 19.1.2024

Camus, A. 5.5.2022. JavaScript Library vs JavaScript Frameworks - The Differences. Blogipostaus. Luettavissa: https://www.microverse.org/blog/javascript-library-vs-javascript-frameworks-the-differences. Luettu: 15.1.2024.

Chim, N. 24.9.2020. JWT Signing Algorithms. Blogi. Luettavissa: https://www.loginradius.com/blog/engineering/jwt-signing-algorithms/. Luettu 15.1.2024.

Date. 2019. Database Design and Relational Theory. Apress. E-kirja. Luettu: 19.1.2024

Ferreira, N. M. 2.11.2023. Color psychology: How color meanings affect your brand. Blogi. Luettavissa: https://www.oberlo.com/blog/color-psychology-color-meanings. Luettu 15.1.2024.

Hakatemia. 2022. JWT Hyökkäykset: JWT Tokenit. Verkkokurssi. Luettavissa: https://www.hakatemia.fi/courses/jwt-hyokkaykset/jwt-tokenit. Luettu: 15.1.2024.

Heroku Dev Center. 2023. The Heroku CLI. Luettavissa: https://devcenter.heroku.com/articles/heroku-cli. Luettu 19.1.2024

Heroku Dev Center. 2023. Deploying with Git. Luettavissa: https://devcenter.heroku.com/articles/git. Luettu 19.1.2024

Holden, S., Martelli, A., Martelli Ravenscroft, A. & McGuire, P. 2023. Python in a Nutshell. 4. painos. O'Reilly Media. E-kirja. Luettu: 19.1.2024

Karjalainen, A. 22.9.2021. Panosta sisällöntuotantoon ja menesty mainonnassasi. Miami Performance Agency blogi. Luettavissa: https://miamiagency.fi/panosta-sisallontuotantoon-ja-menesty-mainonnassasi/. Luettu: 19.1.2024

Khan, S. 15.3.2021. Shahzad Khan, Author at General Assembly Blog. Blogi. Luettavissa: https://generalassemb.ly/blog/author/shahzad-khan/. Luettu: 15.1.2024.

Koivunen, K. 14.9.2018. Minäkö Brändi? Henkilöbrändäyksen abc. Webinaari. Katsottavissa: https://www.youtube.com/watch?v=1uZesJrFHyM&t=1s. Katsottu: 19.1.2024

Kuusimäki, S. 5.1.2023. Sisällöntuotannon ja sisältömarkkinoinnin avulla pääset kätevästi asiakasta palvelevaan sisältöön. Blogi. Luettavissa: https://www.sannakuusimaki.com /sisallontuotanto-sisaltomarkkinointi-palvelee-asiakkaita-ja-hakukoneita. Luettu: 15.1.2024

McEven, M. 16.2.2021. The best JavaScript date libraries in 2021. Blogi. Luettavissa: https://www.skypack.dev/blog/2021/02/the-best-javascript-date-libraries/. Luettu: 15.1.2024.

Mockaroo s.a. Verkkosivu testidatan luomiselle. Luettavissa: https://mockaroo.com/. Luettu 15.1.2024.

MongoDB s. a. MongoDB Atlas: The multi-cloud developer data platform. Luettavissa: https://www.mongodb.com/atlas. Luettu: 15.1.2024.

Motor 3.3.2 documentation s.a. Tutorial: Using Motor With asyncio. Luettavissa: https://motor.readthedocs.io/en/stable/tutorial-asyncio.html. Luettu 19.1.2024.

Nicoara, R. 2023. How to be a Web Developer: A Complete Beginner's Guide on What to Know and Where to Start. Apress L. P. USA. E-Kirja. Luettu: 15.1.2024.

Nygard, M. T. 2018. Release it! Design and deploy production-ready software. 2. painos. Pragmatic Bookshelf. E-kirja. Luettu: 15.1.2024.

PassLib s. a. Dokumentaatio. Luettavissa: https://passlib.readthedocs.io/en/stable/. Luettu: 15.1.2024

PyJWT s. a. Dokumentaatio. Luettavissa: https://pyjwt.readthedocs.io/en/latest/. Luettu: 15.1.2024

Rodriguez, A. 8.2.2015. Introduction to RESTful Web services. IBM blogi. Luettavissa: https://developer.ibm.com/articles/ws-restful/. Luettu 19.1.2024.

Roldán, C. S. 2023. React 18 Design Patterns and Best Practices. 4. painos. Packt Publishing. E-Kirja. Luettu: 15.1.2024.

Roy, S., Daniel, C. & Agrawal, M. 2023. Fundamentals of information technology. Digital Commons @ University of South Florida. E-Kirja. Luettu: 19.1.2024.

Selejan, O., Muresanu, D. F., Popa, L., Muresanu-Oloeriu, I., Iudean, D., Buzoianu, A. & Suciu, S. 2016. Credibility judgments in web page design - a brief review. Journal of medicine and life, 9, 2, s. 115–119.

Stallman, R. 2021. Why Open Source Misses the Point of Free Software. Luettavissa: https://www.gnu.org/philosophy/open-source-misses-the-point.html. Luettu: 19.1.2024.

Uotila, T. 2021. Tietokone Työvälineenä Osa 2 - Versionhallinta: Git ja Github. Helsingin yliopiston verkkokurssi. Luettavissa: https://tkt-lapio.github.io/git/. Luettu 15.1.2024

Uvicorn. 2022. Introduction. Luettavissa: https://www.uvicorn.org/. Luettu 24.1.2024.

Vahtola, M. 2020. Intohimona brändit: Kolme vuosikymmentä brändien parissa. Jyväskylä: Docendo.

Venermo, A. 21.2.2022. Call to Action (CTA) suomeksi ja parhaat vinkit käyttöön. Folcan blogi. Luettavissa: https://folcan.fi/call-to-action-suomeksi/. Luettu: 19.1.2024

W3 Schools s.a. Python Introduction. Verkkotutoriaali. Luettavissa: https://www.w3schools.com/python/python_intro.asp. Luettu 19.1.2024

Österlund, M. 28.2.2022. Vaihtoehtoiset tekstit – ikkuna näkevien maailmaan. Selko digital blogi. Luettavissa: https://selkodigital.fi/vaihtoehtoiset-tekstit-ikkuna-nakevien-maailmaan/. Luettu 19.1.2024.

Österlund, M. 25.10.2022. Saavutettava verkkokauppa on toimiva verkkokauppa. Selko digital blogi. Luettavissa: https://selkodigital.fi/saavutettava-verkkokauppa-on-toimiva-verkkokauppa-2/. Luettu 19.1.2024.