Projektinhallintaohjelmiston suunnitteleminen ja kehittäminen MERN-stack toteutuksena

Jere Järvinen

2022 Laurea

Laurea-ammattikorkeakoulu

Projektinhallintaohjelmiston suunnitteleminen ja kehittäminen MERN-stack toteutuksena

Jere Järvinen

Tietojenkäsittelyn koulutus

Opinnäytetyö

Huhtikuu, 2022

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutus
Tradenomi (AMK)

Jere Järvinen

Projektinhallintaohjelmiston suunnitteleminen ja kehittäminen MERN-stack toteutuksena

Vuosi 2022 Sivumäärä 355

Kaikki ihmiset harjoittavat jokapäiväisessä elämässään projektiluontoista toimintaa, tapahtui se sitten henkilökohtaisessa elämässä vaikkapa muuttolaatikoiden pakkaamisen tai joululahjojen etsimisen tai työelämässä asiakkaalle tehtävän kylpyhuoneremontin tai ohjelmistoratkaisun myynnin muodossa. Koneet soveltuvat erinomaisesti liukuhihnamaisten, toistuvien prosessien tekemiseen, mutta ihmisen vahvuus piilee siinä, kuinka hyvin me pystymme työskentelemään ainutlaatuisten, mukautuvien ja tiettyyn päämäärään pyrkivien työtehtävien, toisin sanoen projektien parissa.

Projektien muodostuessa riittävän monimutkaisiksi on niiden hallinta ihmisellekin haastavaa. Tähän ongelmaan on aikojen saatossa keitetty lukuisia ratkaisuja erilaisista lähtökohdista erityisesti ohjelmistojen muodossa. Useimmat ratkaisut keskittyvät erityisesti suurten organisaatioiden raskaiden projektien hallintaan. Tämä on ymmärrettävää, sillä projektinhallinnan tarve kasvaa sitä mukaa kun mukana olevien ihmisten määrä ja projektien monimutkaisuus kasvaa. Lisäksi kaupallisesta näkökulmasta niissä piilee ohjelmistonpalvelun tarjoajalle suurimmat mahdollisuudet.

Tässä työssä avaan näkökulmia skaalan toisesta päästä, eli henkilökohtaiseen sekä pienten tiimien ja organisaatioiden projektinhallintaan liittyen. Työn keskiössä on ohjelmistoratkaisun suunnittelu ja kehittäminen nimenomaan tämän kohderyhmän tarpeisiin. Ohjelmisto toteutetaan verkkosovelluksena hyödyntämällä MERN-ohjelmistopinon teknologioita. Työtä ei toteuteta yhteistyössä minkään organisaation kanssa, mutta lopputuloksen hyödyntäminen kaupallisessa mielessä tai ilman on mahdollista tulevaisuudessa. Motivaattorina on erityisesti henkilökohtainen tarve ja kiinnostus kyseistä aihetta kohtaan, sekä kehittyminen teknisten ratkaisujen rakentamisessa.

Asiasanat: projektinhallinta, ohjelmisto, MongoDB, Node.js, Express.js, React

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor of Business Administration (BBA)

Jere Järvinen

Planning and building a project management software as a MERN-stack implementation

Year 2022 Pages 355

Every individual engages in project-like activities in their daily life. This might happen in one's personal life in the form of packing boxes when moving house or finding Christmas presents for one's relatives, as well as in a professional context in the form of a bathroom renovation or selling software to a company. Machines are well suited for executing repetitive tasks, but our strength as humans lies in how we are able to work on unique, ever-changing tasks that have a set end-goal, i.e., projects.

As projects become more complex, managing them becomes challenging even for humans. There have been numerous attempts to solve this problem over the years, especially in the form of software solutions. Most of these solutions are developed for the needs of large projects and large organizations. This is understandable, since as the complexity of the project and the number of people participating in it increases, so does the need for managing it. Large organizations also offer the greatest business opportunities for the developers of such solutions.

This thesis project examines the other end of the project spectrum, which consists of personal and small team projects. The main goal of the thesis was to plan and create a software solution prototype for the needs of this target group. The software will be implemented as a web application, utilizing the technologies of the MERN-stack. The thesis project and the software are not made in cooperation with any organization, but commercial use of the end product is possible in the future. The main motivation for building the software is a personal need and interest towards the subject matter, as well as the desire to improve technical skills in implementing such solutions.

Keywords: project management, software, MongoDB, Node.js, Express.js, React

Sisällys

1 Johdanto

1.1 Tavoite

Tämän opinnäytetyön tavoitteena on kehittää ohjelmistoratkaisu erityisesti pienten tiimien ja yksittäisten käyttäjien tarpeisiin. Lopputuloksena syntyvä ohjelmiston prototyyppi on toimiva kokonaisuus, jota on mahdollista jatkokehittää tulevaisuudessa. Osaltaan työn tavoitteina ovat myös projektinhallinnan tarpeiden määrittäminen kyseisen ohjelmiston kohderyhmässä, tehtyjen ratkaisujen perusteleminen sekä toteuttajan teknisen tietotaidon kehittäminen suunnittelu- ja kehitysprosesseissa. Työtä ei toteuteta minkään ulkopuolisen tahon toimeksiannosta, vaan sen taustalla oli ajatus lopputuloksen hyödyntämisestä mahdollisen oman yrityksen tuotteen pohjana tulevaisuudessa.

1.2 Rakenne

Teoriaosuudessa käsittelen projektinhallintaa yleisesti, sen osa-alueita sekä siihen sovellettavia menetelmiä. Otan tarkasteluun myös projektinhallinnan ohjelmistot sekä sen, miten projektinhallinta ilmenee pienten tiimien ja henkilökohtaisen käytön kontekstissa. Teoriaosuus sisältää lisäksi työssä toteutettavan ohjelmiston teknologiaratkaisujen esittelyn. Avaan sen yhteydessä tarkemmin valittujen ratkaisujen toimintaa ja perustelen niiden valintaa.

Seuraavissa kappaleissa käsitellään varsinaisen ohjelmiston suunnitteluprosessia rakenteen ja ominaisuuksien osalta sekä teknistä toteutusta tietokantaratkaisujen, back-endin sekä käyttöliittymän osalta. Näiden yhteydessä nostan esiin myös ohjelmiston keskeisimmät jatkokehityskohteet.

2 Projektinhallinta

Tässä osiossa käsittelen projektinhallinnan teoriaa sekä sen tarpeita pienten tiimien sekä yksittäisen käyttäjän näkökulmasta. Pyrin selvittämään mitkä ovat projektinhallinnan pääasialliset haasteet, ja millä tavoilla ohjelmistotyökalut auttavat näiden haasteiden kanssa.

Projekti-käsite pitää sisällään valtavan määrän erilaisia kokonaisuuksia. Miljardeja maksavan avaruusaluksen suunnittelu kymmenen vuoden aikana on projekti siinä missä huonekalujen järjesteleminen uudessa kodissa muuton jäljiltä. Projektissa voi olla osallisena tuhansia ihmisiä tai yksi ihminen, sen budjetti voi olla miljoonia tai nolla, se voi kestää vuosia tai tunteja ja se voi olla virallisella sopimuksella määritelty tai henkilökohtaiseen harrastukseen liittyvä.

Näistä suurista eroavaisuuksista huolimatta kaikilla projekteilla on tietyt ominaispiirteet, jotka tekevät siitä projektin. (Portny, 2013, s. 11-12)

Projektin tärkein ominaispiirre sen ainutlaatuisuus, sillä ainoastaan se erottaa projektityön liukuhihnamaisesta työstä. (Dinsmore & Cabanis-Brewin, 2014, s. 20) Projekti on itsenäinen kokonaisuus, jonka tavoitteena on määritelty lopputulos. Kyseinen lopputulos on ainoa syy projektin olemassaololle ja sen perusteella arvioidaan projektin onnistuminen. Projektiin liittyy erilaisia rajauksia mm. aikataulun, budjetin, henkilöstön tai muiden resurssien suhteen. Mikäli projektille ei ole esimerkiksi asetettu aikarajaa, ei sitä voida määritelmällisesti kutsua projektiksi. (Portny, 2013, s. 10-11) Lisäksi, projekti itsessään ei ole aktiviteetti, vaan se koostuu aina joukosta aktiviteetteja, joilla on oma elinkaarensa. Usein kyseiset aktiviteetit ovat myös toisistaan riippuvaisia. (Dinsmore & Cabanis-Brewin, 2014, s. 20)

2.1 Projektinhallinnan elinkaari

Vaikka jokainen projekti on lähtökohtaisesti jollain tapaa uniikki kokonaisuus, projektinhallinnan elinkaari sisältää yleisesti samat vaiheet projektista riippumatta. Nämä elinkaaren vaiheet ovat:

- Aloitus
    - Projektin tavoitteen ja sidosryhmien määrittely. Mahdollinen toimeksianto tai sopimus.
- Suunnittelu
    - Pitää sisällään mm. projektin tuotosten, rajausten, työmäärän ja resurssitarpeiden määrittelyn, aktiviteettien suunnittelun ja aikataulutuksen. Riippuen käytettävästä projektinhallinnan menetelmästä suunnittelu voi olla etupainotteista tai tapahtua iteratiivisesti projektin toteutuksen edetessä.
- Toteutus
    - Suunniteltujen aktiviteettien läpivieminen. Projektitiimin kasaaminen, kehittäminen ja johtaminen. Tiedon jakaminen tiimin kesken ja sidosryhmien kanssa.
- Seuranta ja valvonta
    - Projektin rajauksen, budjetin, laadun ja riskien hallinta. Mukautuminen mahdollisiin muutoksiin. Tapahtuu samanaikaisesti toteutusvaiheen kanssa.
- Päättäminen
    - Lopputuloksen hyväksyminen ja mahdollisten virallisten sopimusten päättäminen.

(Phillips, 2011, kappale 1)

2.2 Projektinhallinnan osa-alueet

Projektinhallinta on laaja käsite, joka pitää sisällään useita osa-alueita (ks. Kuvio 1). Ensimmäinen osa-alue on projektin laajuuden ja rajausten määrittely. Projektin onnistuminen edellyttää, että sen puitteissa tehtävä työ, sekä sen lopputulos on tarkasti määritelty. On myös tärkeää asettaa projektille rajaukset, jotta sen mittakaava ei laajene projektin edetessä. Ajankäytön hallinta on merkittävä osa projektinhallintaa, sillä se pitää sisällään projektiin tarvittavien aikaresurssien arvioimisen sekä sen sisältämien aktiviteettien aikatauluttamisen. Projekti sisältää usein kustannuksia, joiden hallitseminen on myös huomioitava. Kustannusten hallinnalla tarkoitetaan projektin kustannusten arvioimista, budjetin hankkimista ja kulujen seurantaa. Ajan ja kustannusten kontrolloimisen vastapainona on pidettävä huoli myös projektin laadun hallinnasta. Halutun lopputuloksen aikaansaamiseksi on projektin lopputulokselle syytä asettaa laatuvaatimukset. Laadunhallintaan kuuluu myös laadun kehittymisen seuraaminen projektin edetessä. (Dinsmore & Cabanis-Brewin, 2014, s. 25-26)

Mikäli projekti toteutetaan tiiminä, henkilöstönhallinta tulee niin ikään osaksi kokonaisuutta. Tähän kuuluu kaikki tekeminen, jolla pyritään mahdollistamaan ja tehostamaan projektitiimin tai siihen muuten osallistuvien tahojen toimintaa. Tästä muutamia esimerkkejä ovat mm. tiimin rakentaminen ja motivoiminen, konfliktien ratkominen sekä erinäiset hallinnolliset tehtävät. Kommunikaation hallinta muodostaa kehyksen sille, miten projektin puitteissa viestitään tiimin sisällä sekä esimerkiksi ylemmän johdon, asiakkaan tai muiden sidosryhmien kanssa. Riskienhallinnan rooli on projektissa mahdollisten riskien tunnistamisessa, niiden ehkäisyssä ja niihin varautumisessa. Nykyinen käsitys projektinhallinnasta ulottuu pelkän projektitiimin sekä -päällikön ulkopuolelle ja näiden ulkopuolisten tahojen tunnistaminen on olennainen osa projektin hallinnan kokonaisuutta. Näin ollen sidosryhmien hallinta lasketaan myös osaksi projektinhallintaa. Sidosryhmillä tarkoitetaan mm. esimiehiä, asiakkaita, loppukäyttäjiä, palveluntarjoajia ja yhteisöjä joihin projekti vaikuttaa tai joiden vuorovaikutusta tarvitaan projektin läpiviemiseksi. (Dinsmore & Cabanis-Brewin, 2014, s. 26-28)

Kuvio 1: Projektinhallinnan osa-alueet

2.3 Projektinhallinnan menetelmät

Projektinhallintaan on olemassa erilaisia malleja. Perinteinen malli, jota nykyisin nimitetään usein myös vesiputousmalliksi, on luonteeltaan lineaarinen ja siinä projektin elinkaari etenee samaa tahtia itse tuotoksen elinkaaren kanssa. Jokainen vaihe käsitellään loppuun ennen siirtymistä seuraavaan vaiheeseen. (Dinsmore & Cabanis-Brewin, 2014, s. 21–22) Vesiputousmalli nojaa vahvasti koko prosessin suunnitteluun ennalta ja välitavoitteiden asettamiseen. Mallin ongelmana on sen joustamattomuus ongelmatilanteissa tai projektin vaatimusten muuttuessa. (Olson, 2020, s. 64)

Vesiputousmallin vastakohtana voidaan pitää Agileksi kutsuttua, iteratiivista mallia. Etupainotteisen suunnittelun sijaan Agilessa työskentely on jaettu lyhyisiin, yleensä korkeintaan muutaman viikon aikajaksoihin (sprint), joiden lopputuloksena syntyy projektia eteenpäin vieviä tuotoksia. Tällä lähestymistavalla saavutetaan läpinäkyvyyttä projektin edistymiseen, sekä ketteryyttä ongelmien tai uusien tarpeiden ilmaantuessa. (Wells & Kloppenborg, 2018, s. 8-9) Agile on vahvimmillaan erityisesti pienten, itseohjautuvien tiimien käytössä. Malli on saavuttanut vahvan jalansijan erityisesti ohjelmistoalalla, jossa uusien ominaisuuksien toteuttaminen nopealla kierrolla on erityisen tärkeää. (Olson, 2020, s. 66)

Projektin elinkaaren hallintaa voidaan virtaviivaistaa entisestään käyttämällä inkrementaalista, prototyyppeihin perustuvaa lähestymistapaa. Tämä tarkoittaa sitä, että asiakkaan käytettäväksi pyritään tuottamaan ensin vähimmäisvaatimukset täyttävä tuotos. Tätä kautta saadaan uutta tietoa, jonka perusteella tuotteen vaatimukset mukautuvat ja projektin edetessä tuotetta kehitetään pienin askelein kohti lopullista vaatimustasoa. (Dinsmore & Cabanis-Brewin, 2014, s. 22) Prototyyppiajattelu yhdistetään usein Agile-malliin, sillä tuomalla sprinttien tuotokset nopeasti asiakkaan arvioitavaksi, saadaan tietoa tarvittavista muutoksista tulevien sprinttien pohjaksi. (Olson, 2020, s. 65)

2.4 Projektinhallinnan ohjelmistot

Projektien hallintaan voidaan käyttää joko joukkoa erillisiä, tiettyihin asioihin erikoistuneita ohjelmistoja tai ns. integroitua projektinhallintaohjelmistoa. Ensin mainittu lähestymistapa tarkoittaa sitä, että esimerkiksi aikataulutukseen, tiimin kommunikaatioon, tiedon tallentamiseen ja tehtävien kirjaamiseen käytetään omia ohjelmistojaan. Tässä tapauksessa yksittäisiin toimintoihin on käytössä yleensä tehokkaat ja tutut työkalut, mutta projektin kokonaisuuden hahmottamisesta ja datan ylläpitämisestä tulee helposti työlästä. Integroidulla projektinhallintaohjelmistolla tarkoitetaan ohjelmistoa, joka yhdistää tärkeimmät projektinhallintaan tarvittavat toiminnot samaan pakettiin. Tällainen ohjelmisto saattaa vaatia käyttäjien erillistä koulutusta, mutta eri toimintojen linkittyneisyys helpottaa useita toimintoja. (Portny, 2013, s. 330–333)

Projektinhallinnan ohjelmistojen tarkoituksena on yleisesti tarjota ratkaisuja mm. seuraaviin ongelmakohtiin.

- Aikataulutus
    - Projektiin liittyvien tehtävien keston määrittäminen
    - Henkilöresurssien hallinta
- Projektiin liittyvän tiedon hallinta
    - Tehtävien läpinäkyvyys
    - Tiedot mm. mahdollisista riskeistä ja työtaakan muutoksista
    - Kustannusten läpinäkyvyys
- Projektin tuotosten ja edistymisen raportointi
- Kommunikaatio ja yhteistyö
- Tuottavuuden mittaaminen
- Laadunvarmistus

(Kundu, Bishoi, Bhattacharya & Chowdhury, 2015, 1-2)

Projektinhallinnan ohjelmistoja on olemassa runsaasti ja niitä on kehitetty useista eri näkökulmista. Kaikki niistä eivät luonnollisesti ole parhaimmillaan kaikissa käyttötapauksissa. Osa on esimerkiksi erikoistunut vain agile-mallin mukaiseen projektinhallintaan tai suurten organisaatioiden tarpeisiin. Alan merkittäviä toimijoita ovat mm. JIRA, Microsoft Project, Clickup, Monday.com, Notion ja Trello. Näistä erityisen kiinnostavia ratkaisuja ovat Clickup, Notion ja Trello, sillä ne ovat lähtökohdiltaan lähimpänä tässä työssä toteutettavaa sovellusta. Edellä mainittuja ohjelmistoja on mahdollista käyttää kevyellä ominaisuusvalikoimalla, ja ne soveltuvat hyvin henkilökohtaisten- tai pienten tiimiprojektien hallintaan.

2.5 Pienten projektien hallinta

Tämän työn kontekstissa projektinhallinnan teorian osalta on erityisen kiinnostavaa se, minkälaisia projektinhallinnan tarpeet ovat pienten projektien puitteissa.

Projektien hallintaan on olemassa monia parhaita käytänteitä, joiden noudattaminen on hyödyksi projektin koosta riippumatta. Kaikki suurten projektien yhteydessä sovellettavat käytänteet eivät kuitenkaan ole helposti käännettävissä pienten projektien tarpeisiin. Projektin määritteleminen ”pieneksi” on luonnollisesti hyvin tulkinnanvaraista. Rowe (2007) esittää mm. seuraavanlaisia määritelmiä pienelle projektille, jotka ovat käyttökelpoisia tässä kontekstissa:

- projektin kesto on korkeintaan 6 kuukautta
- työtä ei tehdä pääsääntöisesti kokopäiväisesti
- projektitiimin koko korkeintaan 10 henkilöä

- vaatii vain pienen määrän erilaisia osaamisalueita
- projektilla on vain yksi päätöksentekijä

(Rowe, 2007)

Erityisen tärkeää on tietää, mitkä ominaisuudet ovat tämän kaltaiselle kohderyhmälle relevantteja projektinhallinnan ohjelmistossa. Kyseisen kohderyhmän projekteja ovat esimerkiksi

- Opinnäytetyön tekeminen
- Ravintolan verkkosivujen rakentaminen
- Urheiluseuran varainkeruutapahtuman suunnittelu
- Opiskelijoiden ryhmätyön toteuttaminen
- Pienen urakoitsijan remonttiprojekti
- PK-yrityksen markkinointikampanjan suunnittelu

Pienissä organisaatioissa jokaisella projektilla ei yleensä ole erillistä päällikköä, vaan yksi henkilö voi olla vastuussa lukuisten projektien johtamisesta ja seurannasta. (Kerzner & Kerzner, 2013, s. 409) Projektinhallinnan ohjelmistot ovat omiaan auttamaan kokonaisuuksien hahmottamisessa tällaisessa tilanteessa.

Tyypillinen ongelma pienissä projekteissa on usein suunnittelun puute ennen projektin aloittamista. Vaikka kokonaistyömäärä olisi lähtökohtaisesti pieni, on vaarana, että ilman suunnittelua projektin rajaus tulee muuttumaan sen edetessä tai kriittisiä asioita jää huomioimatta. Suunnittelussa on tärkeää välttää liiallista yksityiskohtiin menemistä ja osallistaa henkilöt, jotka tulevat tekemään varsinaisen työn. (Rowe, 2007)

Westcott (2004) nostaa esiin pienten projektien hallintaan käytettäviä tekniikoita ja malleja. Näistä esimerkeistä käy ilmi kaikenkokoisten projektien yhteinen tarve kokonaisuuden pilkkomiseen. Kaikkein pienimpienkin projektien hallinnassa olennaista on tehtävien kirjaaminen ja niiden aikatauluttaminen. Tehtäviin liittyen tarpeita ovat mm. aikatauluttaminen, vastuuhenkilöiden määrittäminen, keskinäisten riippuvuuksien esittäminen, sekä budjetointi. Projektin lopputuloksen määrittely ja rajaus, sekä toteutuksen kuvaaminen sanallisesti ovat myös vaadittuja ominaisuuksia pienten projektien hallintaan käytettävässä ratkaisussa. (Westcott, 2004, s. 155-160)

Rowen (2007) mukaan pienten projektien yhteydessä relevantteja toimenpiteitä ovat:

- Projektin alkumäärittely, sisältäen mm.
    - tiimin jäsenet sekä roolit ja vastuut
    - asiakkaan ja muut sidosryhmät
    - projektin kuvauksen, budjetin, tavoitteet ja rajauksen

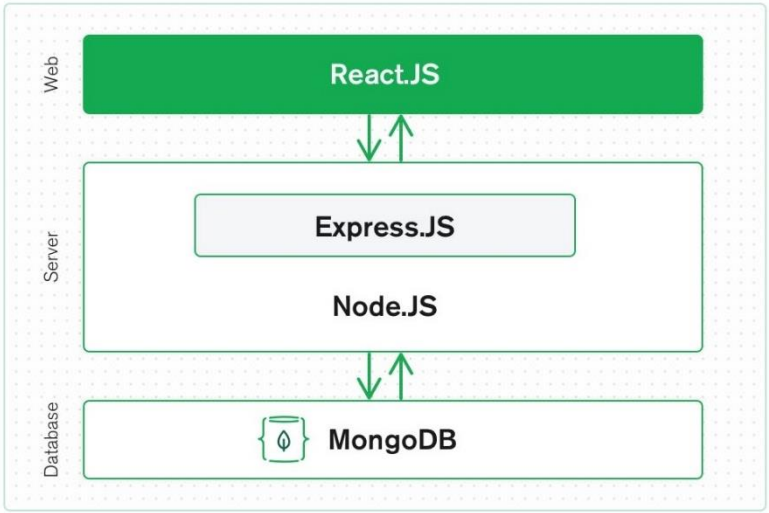
    - ulkoiset riippuvuudet ja tarkan lopputuloksen määrittelyn
- Työn pilkkominen pienempiin kokonaisuuksiin ja niiden lopputulosten määrittely
- Konkreettisen tehtävälistan laatiminen
- Aikataulutus
    - Tarpeen ylemmän tason kokonaisuuksien tasolla, mutta ei aina tehtävätasolla
- Katsaus riskeihin ja suunnitelmat niiden varalle

3 MERN-stack

Stack-sanaa käytetään tietotekniikassa kuvaamaan tiettyjen ohjelmointikielten, ohjelmistokehysten, sekä palveluiden muodostamaa kokonaisuutta, jotka yhdessä muodostavat edellytykset kokonaisvaltaisten ohjelmistotuotteiden kehittämiseen. Tyypillisesti stack sisältää ratkaisut tietokannan hallintaan, backend-kehitykseen, sekä frontend-kehitykseen.

MERN-stack koostuu seuraavasta neljästä osa-alueesta:

- **MongoDB** - Tietokanta
- **Express.js** – Backend (palvelinsovellus)
- **React** – Frontend
- **Node.js** – Backend (palvelinsovelluksen alusta)



Kuvio 2: MERN-stack (MongoDB. 2021b)

Yksi MERN-stackin suurimmista vahvuuksista on sen osa-alueiden vahva asema suosittuina vapaan lähdekoodin ratkaisuina, mikä takaa relevanssin pitkällä aikavälillä ilman huolta teknologioiden vanhenemisesta. Toinen merkittävä vahvuus MERN-stackissa on siinä kautta linjan käytetty JavaScript-ohjelmointikieli. Oppimiskynnyksen ja kehitystyön kannalta yleisesti on hyödyllistä, että koko stackin käyttämiseen riittää yhden ohjelmointikielen osaaminen. (Hoque, 2020, s. 17)

3.1.1 JavaScript

JavaScript on alustasta riippumaton olio-ohjelmointikieli, jota voidaan käyttää joko asiakas- (client) tai palvelinsovelluksissa (server). Client-puolella JavaScriptia käytetään verkkosivujen dynaamisen ja interaktiivisen sisällön luomisessa, kun taas palvelinpuolella kokonaisvaltaisten back-end sovellusten rakentaminen on mahdollista JavaScript-ajoympäristöjen, kuten Node.js, avulla. (developer.mozilla.org, 2022a)

Tarkemmin määriteltynä JavaScript on ns. komentokieli, eli ohjelmointikieli, jonka sisältö tulkataan reaaliajassa ajoympäristössä. Tämä tarkoittaa, että komentokielen koodille ei tarvita erillistä käännösvaihetta ennen suorittamista. Komentokielet ovat yleisiä nimenomaan web-kehityksessä. (geeksforgeeks.org, 2022)

3.1.2 MongoDB

Tietokantapalvelut voidaan jakaa karkeasti kahteen kategoriaan: SQL- ja NoSQL -tyyppisiin. SQL-tietokannalla tarkoitetaan ns. relationaalista tietokantaa, jossa tieto mallinnetaan taulukkorakenteilla. SQL-taulukko pitää sisällään sarakkeita, joiden sisältämät tietotyypit ovat ennalta määrättyjä. Tietueet esitetään riveinä taulukossa ja niiden tulee noudattaa taulukon skeemaa. Taulukoiden välille voidaan luoda suhteita viittaamalla tietueisiin uniikeilla avaimilla. (MongoDB, 2021b)

Projektit

ID	Otsikko
1	Projekti 1
2	Projekti 2

Käyttäjät

ID	Nimi
1	Käyttäjä 1
2	Käyttäjä 2

Projektin käyttäjät

ID	ProjektiID	KäyttäjäID
1	1	1
2	2	1
3	2	2

Kuvio 3: Esimerkki SQL-tietokannan rakenteesta.

MongoDB on avoimen lähdekoodin NoSQL-tyyppinen tietokantapalvelu. NoSQL-termillä voidaan tarkoittaa mitä tahansa tietokantatyyppiä, jossa dataa käsitellään relationaalisen taulukkorakenteen sijaan muilla tavoilla. (MongoDB. 2021a)

MongoDB:n toimintaperiaate perustuu JavaScript-objektien (JSON) tyylisiin “dokumentteihin” ja niitä sisältäviin kokoelmiin (ks. Kuvio 4). Dokumentit muodostuvat tietokentistä, joiden ei ole pakko noudattaa ennalta määriteltyä skeemaa. Näin ollen tietueiden rakennetta voi muuttaa joustavammin sitä mukaa kun sovelluksen vaatimukset muuttuvat. (Hoque, 2020, s. 15)

Kuvio 4: Esimerkki MongoDB-dokumenteista.

Edellä kuvatussa esimerkissä on kuvattu kaksi erilaista kokoelmaa. Aaltosuluilla rajatut osuudet edustavat yksittäistä dokumenttia. Toisin kuin SQL-taulujen rivit, MongoDB-dokumentit voivat sisältää itsessään lista- tai taulukkomuotoisia kenttiä. Näin ollen projektiin liittyvät käyttäjät voidaan esittää samassa dokumentissa projektin muiden tietojen kanssa.

Dokumenttipohjaisten tietokantojen vahvuudet ovat erityisesti kyselyjen nopeudessa sekä skaalautuvuudessa. Dokumenttipohjaisen tietokannan koko on useimmiten SQL-tietokantaa suurempi, mutta se on helpompi skaalata horisontaalisesti, eli jakaa useamman palvelimen vastuulle. (Fowler, 2015, s. 15-16)

Tämän projektin puitteissa tietokantaratkaisuna käytettiin MongoDB Atlas -nimistä Database-as-a-service -palvelua (DBaaS). DBaaS tarkoittaa palvelua, jossa tietokanta sijaitsee paikallisen fyysisen koneen sijaan palveluntarjoajan pilvessä. Hallitsemalla tietokantaa pilvessä, pystytään sitä skaalaamaan erittäin joustavasti. Tällä saavutetaan kustannussäästöjä, sekä parempaa luotettavuutta loppuasiakkaalle. DBaaS on lisäksi käyttäjän näkökulmasta helppo ratkaisu, sillä se hoitaa useimmat infrastruktuurin toimintaan liittyvät toimenpiteen automaattisesti tämän puolesta. Sovellusprototyyppien luomiseen MongoDB Atlas tarjoaa ilmaiseksi 500mb kokoisen tietokannan. (MongoDB. 2022b)

MongoDB:n yhteydessä käytetään usein apuvälineenä Mongoose-nimistä tietueobjektien mallintamiseen ja hallintaan tarkoitettua Node.js -kirjastoa. Käytännössä Mongoose tarjoaa helpon tavan määritellä tietorakenteita sovellustasolla, huolehtia tietueiden validoinnista sekä tehdä tietokantakyselyjä. (MongoDB. 2022a)

3.1.3 Node.js

Node.js on avoimen lähdekoodin ajoympäristö, joka mahdollistaa backend kehittämisen samalla JavaScript-kielellä, jota käytetään myös frontend-kehityksessä. Noden vahvuutena on erityisesti sen soveltuvuus palvelinkutsujen ”asynkroniseen” käsittelyyn. Tämä tarkoittaa sitä, että ohjelman ei tarvitse jäädä odottamaan tietyn toiminnon valmistumista suorittaakseen seuraavan, vaan useita toimintoja voidaan käsitellä samanaikaisesti. (Oluyege, 2019, s. 10)

Kehittäjän näkökulmasta Noden tekee houkuttelevaksi myös se ympärille muodostunut erittäin laaja ekosysteemi avoimen lähdekoodin kirjastoja ja työkaluja. Ulkoisten kirjastojen käyttäminen tapahtuu Nodessa helposti ns. pakettienhallintajärjestelmien avulla. Oletuksena Noden mukana tuleva järjestelmä on nimeltään Node Package Manager (NPM). Esimerkki Nodessa hyödynnettävistä kirjastoista on Express.js, joka on yksi MERN-stackin kulmakivistä. (Hoque, 2020, s. 13-14)

3.1.4 Express.js

Express.js on Node.js:n päälle rakennettu ohjelmistokehys, joka sisältää toiminnallisuuksia kokonaisten web-sovellusten ja tai pelkkien rajapintojen kehittämiseen. Tämän työn kontekstissa Expressiä hyödynnetään yksinomaan rajapinnan puitteissa. Rajapinnalla tarkoitetaan tässä kontekstissa palvelinsovellusta, joka välittää dataa frontend-sovelluksen ja tietokannan välillä.

Vaikka Express toiminnot ovat toteutettavissa myös Noden natiiveja ominaisuuksia hyödyntämällä, säästää se kehittäjän aikaa vähentämällä tarvetta perustoimintojen manuaaliseen kirjoittamiseen. Näitä toimintoja ovat mm. HTTP-pyyntöjen reititys, pyyntöjen sisällön, URL-parametrien ja evästeiden lukeminen sekä virheiden käsittely. (Mardan, 2018, s. 52)

Expressin toiminta perustuu käytännössä kokonaan ns. välifunktioihin (middleware). Välifunktioissa voidaan käsitellä sovelluksen vastaanottaman pyynnön sisältöä ja sen vastausta, lopettaa pyynnön käsittelyprosessi lähettämällä vastaus tai ketjuttaa se seuraavalle välifunktiolle. Välifunktio ottaa parametreina http-pyyntöobjektin (req), vastausobjektin (res) sekä funktion, jolla pyyntö siirretään eteenpäin (next). (expressjs.com, 2022a)

Välifunktiota voidaan kutsua joko kaikkien pyyntöjen yhteydessä, tai rajatummin pyynnön reitin tai sen http-verbin perusteella (ks. Kuvio 5).

```
const funktio = (req, res, next) => {
    // Middleware-funktio
}

app.use(funktio) // Kaikilla pyynnöillä
app.use("/projekti", funktio) // Kaikilla projekti-reitin pyynnöillä
app.get("/projekti/:id", funktio) // GET-pyynnöillä kyseiseen reittiin
app.post("/projekti", funktio) // POST-pyynnöllä kyseiseen reittiin
```

Kuva 1: Välifunktion käyttö Express-sovelluksessa.

Välifunktiot voidaan jakaa viiteen eri kategoriaan:

- Sovellustason välifunktiot
    - Sidottu suoraan varsinaiseen Express-sovellukseen
- Reititin-tason välifunktiot
    - Sidottu Express Router-instanssiin
- Virheenkäsittelijä -välifunktiot
    - Ottaa tavallisten ”pyyntö”-, ”vastaus”-, ja ”seuraava” -parametrien lisäksi virheen parametrina
- Sisäänrakennetut välifunktiot
    - Express sisältää kolme sisäänrakennettua välifunktiota:
        - *express.static* – staattisten tiedostojen, esim. HTML- tai kuvatiedostojen tarjoamiseen
        - *express.json* – parsii vastaanotettavien JSON-muotoisten pyyntöjen sisällön
        - *express.urlencoded* – parsii pyynnöt, joiden sisältö on sisällytetty pyynnön URL:ään
- Kolmannen osapuolen välifunktiot
    - Node.js moduuleina asennettavia, valmiita välifunktioita usein käytetyille toiminnoille.

(expressjs.com, 2022a)

Laajemmissa Express-sovelluksissa yleisenä käytäntönä on ryhmitellä erityyppisten reittien käsittely erillisten ”reitittimien” (router) vastuulle. Reititin on apuväline monimutkaisten reititysten käsittelyyn ja se voidaan nähdä eräänlaisena pienoissovelluksena varsinaisen Express-sovelluksen sisällä. Reitittimen käsiteltäväksi ohjataan kaikki tietynlaisen reitin pyynnöt. esimerkiksi projekteihin liittyvien pyyntöjen polun ollessa muotoa */projects/…* voidaan kaikki tämän reitin alle osuvat pyynnöt ohjata niistä vastaavalle reitittimelle. Näin sovelluksen

rakenne voidaan pitää modulaarisena ja helpommin ymmärrettävänä. Reititintä käytetään varsinaisessa sovelluksessa välifunktion tapaan (ks. kuva 3). (expressjs.com, 2022b)

```
var express = require('express')
var router = express.Router()

router.use(funktio) // Routeriin sidottu middleware

// Reitit
router.get('/', funktio) // Hae kaikki
router.get('/:id', funktio) // Hae yksi
router.post('/', funktio) // Luo

module.exports = router
```

Kuva 2: Esimerkki Express.js-reitittimestä.

```
var projectsRouter = require("./routes/projects.js")

app.use("/projects", projectsRouter)
```

Kuva 3: Reitittimen käyttö Express-sovelluksessa.

Yksinkertainen esimerkki pyynnön käsittelystä Expressin välifunktioilla:

1. Sovellus ottaa vastaan pyynnön
2. Ensimmäinen välifunktio kirjaa pyynnön lokiin
3. Pyyntö siirtyy reitin perusteella oikealle reitittimelle
4. Käyttäjä tunnistetaan reitittimen sisältämässä välifunktiossa.
    - Mikäli tunnistautuminen onnistui, siirtyy pyyntö varsinaiseen käsittelyyn.
    - Muuten asiakkaalle vastauksena virhe
5. Varsinainen pyynnön käsittely (esim. tietokantakyselyt)
    - Mikäli pyynnön käsittely onnistuu, lähetetään asiakkaalle haluttu vastaus
    - Jos käsittelyssä tapahtuu virhe, välitetään pyyntö virheenkäsittelijä-välifunktiolle, joka antaa asiakkaalle vastauksena virheen

3.1.5 React

React on frontend-kehitykseen tarkoitettu JavaScript-kirjasto, jonka pääperiaatteet ovat käyttöliittymien rakentaminen yleiskäyttöisistä komponenteista sekä käyttöliittymän sisällön dynaaminen päivittyminen sovelluksen tilan perusteella. (Hoque, 2020, s. 16) React-komponenttien vastuulla on heijastaa sovelluksen tila käyttäjän näkyviin selaimessa aina sen

muuttuessa. Toisin sanoen, sovelluksen koodi tuottaa dataa, joka välitetään komponentille. Komponentti muodostaa ja päivittää selaimessa näytettävää sisältöä datan perusteella. (Boduch & Derks, 2020, s. 10)

Kuvio 5: Reactin toimintaperiaate.

Reactin komponentteihin perustuva lähestymistapa sovelluksen pilkkomiseen poikkeaa merkittävästi aikaisemmin vallinneista arkkitehtuurikäytänteistä, erityisesti ns. MVC (Model-View-Controller) -mallista. MVC-mallissa sovelluksen data, näkymän tila ja näkymän hallinta on erotettu toisistaan, kun taas React-komponentti on itsenäinen yksikkö, joka sisältää kaiken tiettyyn käyttöliittymän osaan vaadittavan koodin. (Singhal & Corvalan, 2016, s. 2)

Yksi Reactin erityispiirteistä on sen käyttämä strategia selaimessa näytettävän sisällön manipuloimiseen. Verkkosivun sisällön muuttaminen on perinteisesti tapahtunut suoraan sivun DOM:ia (Document Object Model) päivittämällä. Kyseiset operaatiot ovat kuitenkin erittäin hitaita, joten DOM-päivitysten määrän kasvaessa sivun suorituskyky väistämättä heikkenee. Monissa JavaScript -ohjelmistokehyksissä DOM-päivitysten määrä on usein myös tarpeettoman suuri johtuen niiden toimintaperiaatteesta. Reactin osalta kyseistä ongelmaa on lähdetty ratkaisemaan ns. virtuaalisen DOM:in avulla. Yksinkertaistettuna virtuaalinen DOM on kopio varsinaisesta DOM:sta ilman kykyä päivittää sivun sisältöä, mutta sen manipuloiminen on huomattavasti nopeampaa. React-sovelluksessa jokaisen DOM-päivityksen yhteydessä koko Virtual DOM -puu rakennetaan uudestaan. React vertaa uutta virtuaalista DOM:ia aikaisempaan, etsii muuttuneet osat ja tekee muutokset varsinaiseen DOM:iin vain niiden kohdalla. Näin ollen suorien DOM-manipulaatioiden määrä on aina mahdollisimman pieni ja sivun suorituskyky voidaan pitää korkeana. (Codeacademy.com, 2022)

HTML-elementtien kirjoittamisen helpottamiseksi React sisältää JSX-nimisen JavaScript-syntaksin laajennoksen. Käytännössä JSX mahdollistaa Reactin createElement() -funktioiden kirjoittamisen suoraan HTML:ää muistuttavalla syntaksilla.

```
// Ilman JSX:ää
const element = React.createElement("button", { type: "submit" }, ["This is a button"]);

// JSX-syntaksilla
const elementWithJsx = <button type="submit">This is a button</button>;

// JSX-syntaksi propeilla
const prop1 = "submit";
const prop2 = "This is a button";
const elementWithJsxAndProps = <button type={prop1}>{prop2}</button>;
```

Kuva 4. Esimerkki JSX-syntaksista.

4 Sovelluksen sisältö

Sovelluksen suunnitteluprosessissa ensimmäinen askel oli määrittää lopputuloksen tavoite. Keskeisimmät kysymykset olivat, mihin tarpeeseen sovellus tehdään ja mitkä sen vähimmäisvaatimukset ovat. Sovelluksen kohderyhmää ovat ensisijaisesti pienet, alle 8 henkilön organisaatiot ja tiimit sekä yksittäiset käyttäjät.

Minimivaatimukset sovellukselle ovat seuraavat:

- Luotettava käyttäjän tunnistautuminen (kirjautuminen ja rekisteröityminen)
- Projektien luominen, poistaminen, muokkaaminen ja seuranta yksittäisenä käyttäjänä tai tiimitasolla
- Projektiin liittyvien aktiviteettien hallinta
- Projektien jaottelu rajattuihin konteksteihin (työtilat)
- Projektien jakaminen ja yhteistyö muiden käyttäjien kanssa
- Käyttäjän omien tietojen hallinta

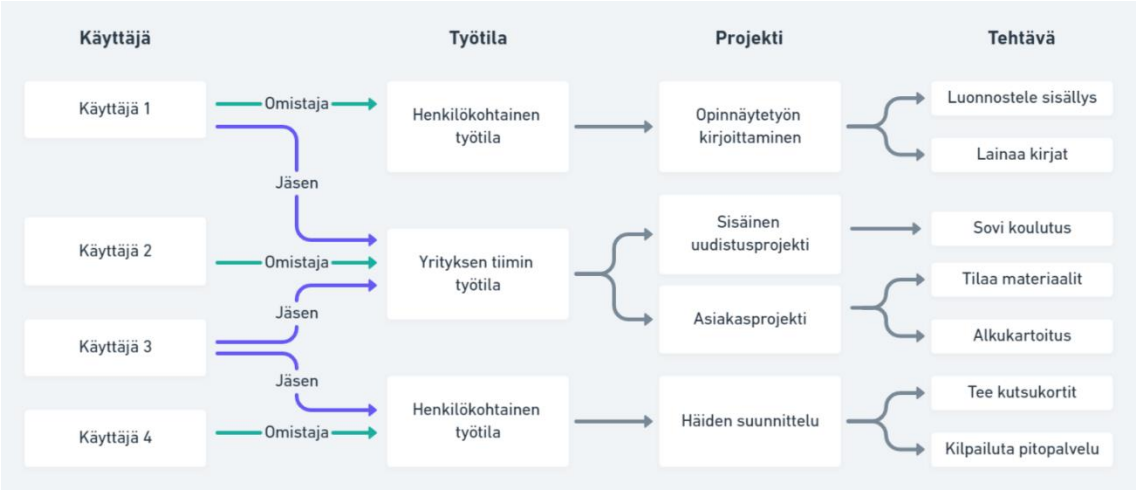
4.1 Rakenne

Sovelluksen rakenteen hahmottelu toteutettiin prosessikaavioiden avulla. Lopullinen suunnitelma rakenteesta muodostui usean iteraation kautta, päätyen lopulta neliportaiseen malliin, joka pitää sisällään käyttäjät, työtilat, projektit ja tehtävät. Varhaisimmissa rakenneluonnoksissa oli mukana mm. organisaatio- ja työntekijä -tasot, mutta nämä pudotettiin pois lopullisesta suunnitelmasta, jotta sovellus pysyisi uskollisena sen varsinaiselle kohderyhmälle sekä käytettävyyden kannalta mahdollisimman helpposelkoisena.

Lopullisessa mallissa käyttöprosessi on hyvin yksinkertainen. Käyttäjätilin luomisen jälkeen käyttäjä luo itselleen työtilan (workspace), joka voi olla tyypiltään kaupallinen (business) tai yksityinen (private). Työtilatyypit eroavat toisistaan toistaiseksi vain muutamien ominaisuuksien osalta, mutta jatkokehityksen kannalta jako on tärkeää, jotta uusia ominaisuuksia

voidaan tuoda valikoidusti käyttöön joko kaupallisessa tai yksityisessä kontekstissa. Työtilan luoja on lähtökohtaisesti sen omistaja. Omistaja pystyy antamaan muille käyttäjille pääsyn työtilaan kutsulinkin kautta. Työtilaan kutsutuilla käyttäjillä (myöh. jäsenet) on määritelty rooli, joka määrittää mitä he pystyvät työtilan sisällä tekemään. Nykyisessä suunnitelmassa rooleja on kaksi: katselurooli (view) ja muokkausrooli (edit). Kutsuttu jäsen on aina aluksi katseluroolilla, mutta työtilan omistaja voi antaa jäsenelle muokkausroolin.

Kaikki projektit ovat sidottuja tiettyyn työtilaan, joten työtilan luomisen jälkeen sen yhteyteen voidaan alkaa luomaan projekteja. Kaupallisessa työtilassa projektille asetetaan tieto siitä, onko kyseessä sisäinen- vai asiakasprojekti. Yksityisessä työtilassa tätä jaottelua ei tehdä. Projekti sisältää monenlaista tietoa tiimin koostumuksesta liitetiedostoihin ja kommentteihin, mutta päivittäisen työskentelyn kannalta olennaisin osa sovelluksesta ovat projekteihin sidotut tehtävät, jotka muodostavat alimman tason sovelluksen rakenteesta. Tehtävä sijaitsee aina tietyn projektin alla ja sille asetetaan erikseen vastuuhenkilö sekä arvioitu valmistumispäivä.



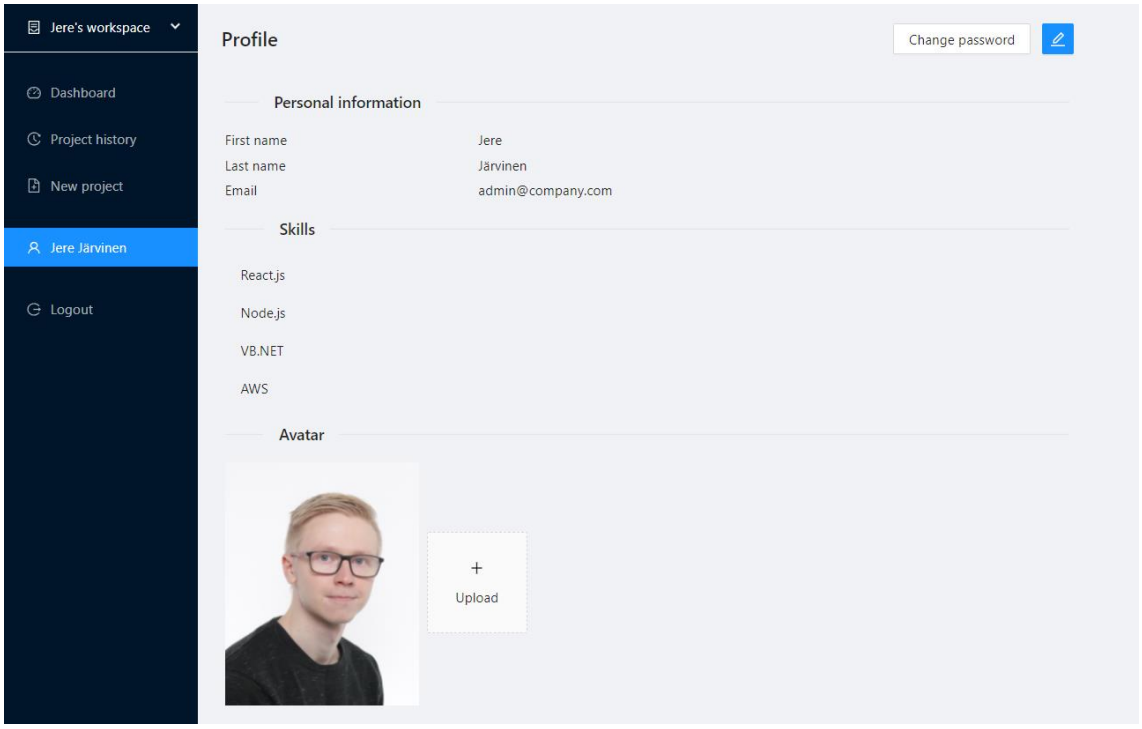
Kuvio 6: Sovelluksen rakenne.

4.2 Ominaisuudet

Sovelluksen perusrakenteen ollessa selvillä, seuraava vaihe suunnittelussa oli haluttujen ominaisuuksien tarkempi määrittely. Varsinaisen prototyypin kehittäminen tapahtui tässä osiossa mainittujen asioiden pohjalta.

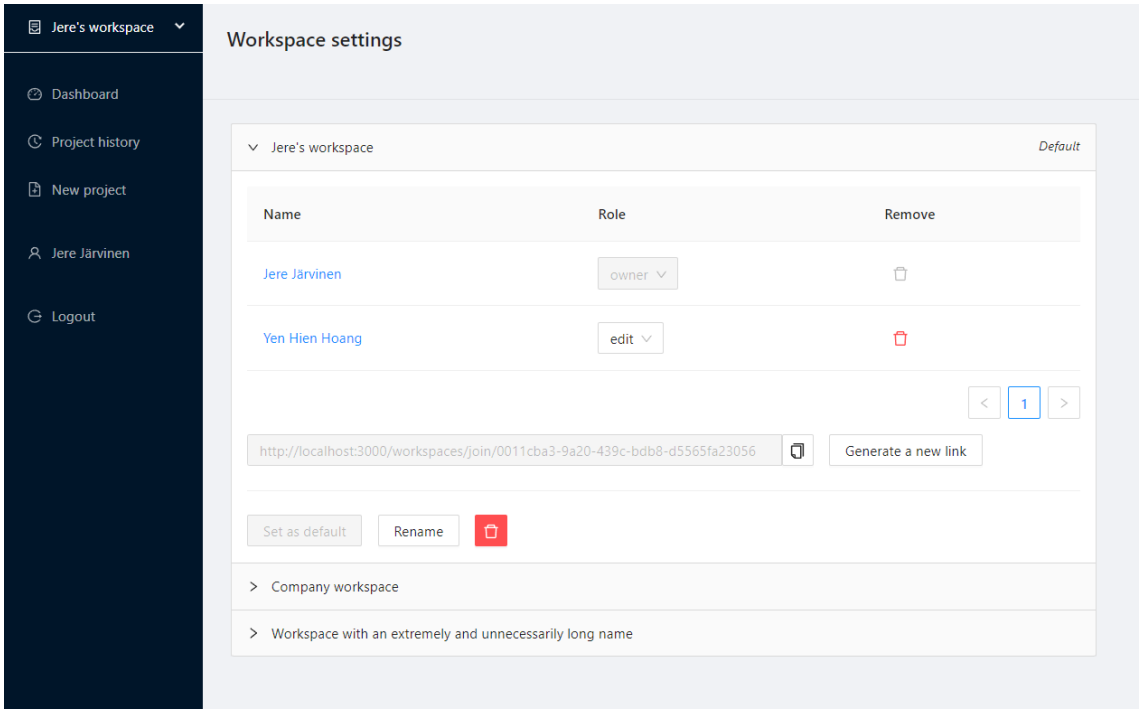
Käyttäjä-tasoon liittyen avainominaisuudet ovat luonnollisesti käyttäjätilin luominen ja hallinta sekä sisään- ja uloskirjautuminen. Ensimmäisessä prototyypissä tilin luomisen yhteyteen ei rakennettu sähköpostivarmistusta. Vaikka varmistus on kriittinen ominaisuus sovellusta julkaistessa, prototyypin rakentaminen onnistuu myös ilman sitä. Käyttäjätilin hallinnassa voidaan muokata käyttäjän nimeä, vaihtaa salasana, lisätä osaamisalueita sekä asettaa ja

poistaa profiilikuva, joka näkyy mm. projektin kommenttien yhteydessä. Lisäksi käyttäjät pystyvät tarkastelemaan muiden työtilan käyttäjien profiileja.



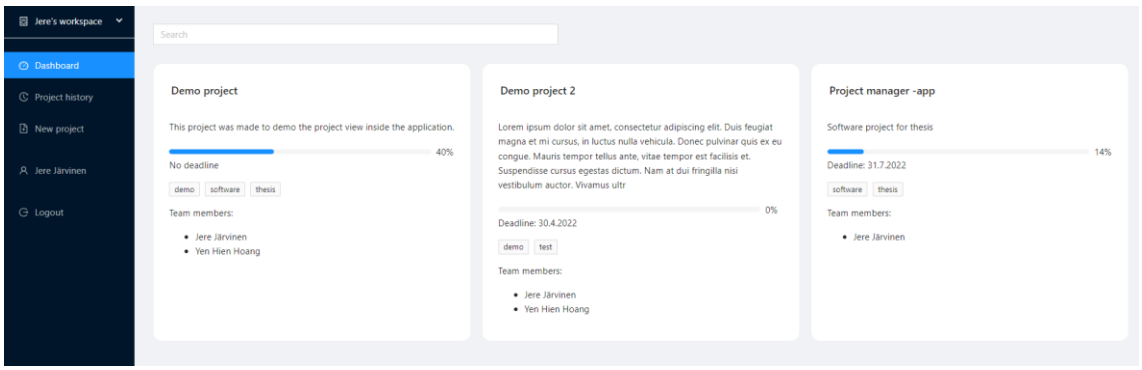
Kuva 5: Käyttäjän profiilisivu.

Työtilojen suhteen toimivaan prototyyppiin vaadittavat vähimmäisominaisuudet ovat työtilojen luominen, poistaminen sekä niiden välillä vaihtaminen. Työtilaan on oltava mahdollista kutsua uusia käyttäjiä kutsulinkin kautta, mutta samalla työtilan omistajan tulee pystyä poistamaan käyttäjiä työtilasta ja muuttamaan näiden rooleja. Työtilojen asetuksista omistaja pystyy muuttamaan tilan nimeä ja kaikki käyttäjät voivat asettaa jonkin heidän käytössään olevista työtiloista oletukseksi. Oletus määrittää sen, mikä työtila käyttäjällä on aktiivisena aina sisäänkirjautumisen jälkeen.



Kuva 6: Työtilojen asetukset.

Sovelluksen tietorakenteen tasoista projekteihin liittyy suunnitelmassa eniten ominaisuuksia. Projektien yleisnäkymänä toimii ns. dashboard-sivu, mikä pitää sisällään kaikki meneillään olevat projektit. Menneet projektit näytetään projektihistoria-sivulla taulukkomuodossa. Dashboardilla projektit näytetään ruudukkomaisesti ”kortteina”, joissa näytetään kaikkein olennaisimmat tiedot. Näkymää voidaan rajata hakukentän avulla.



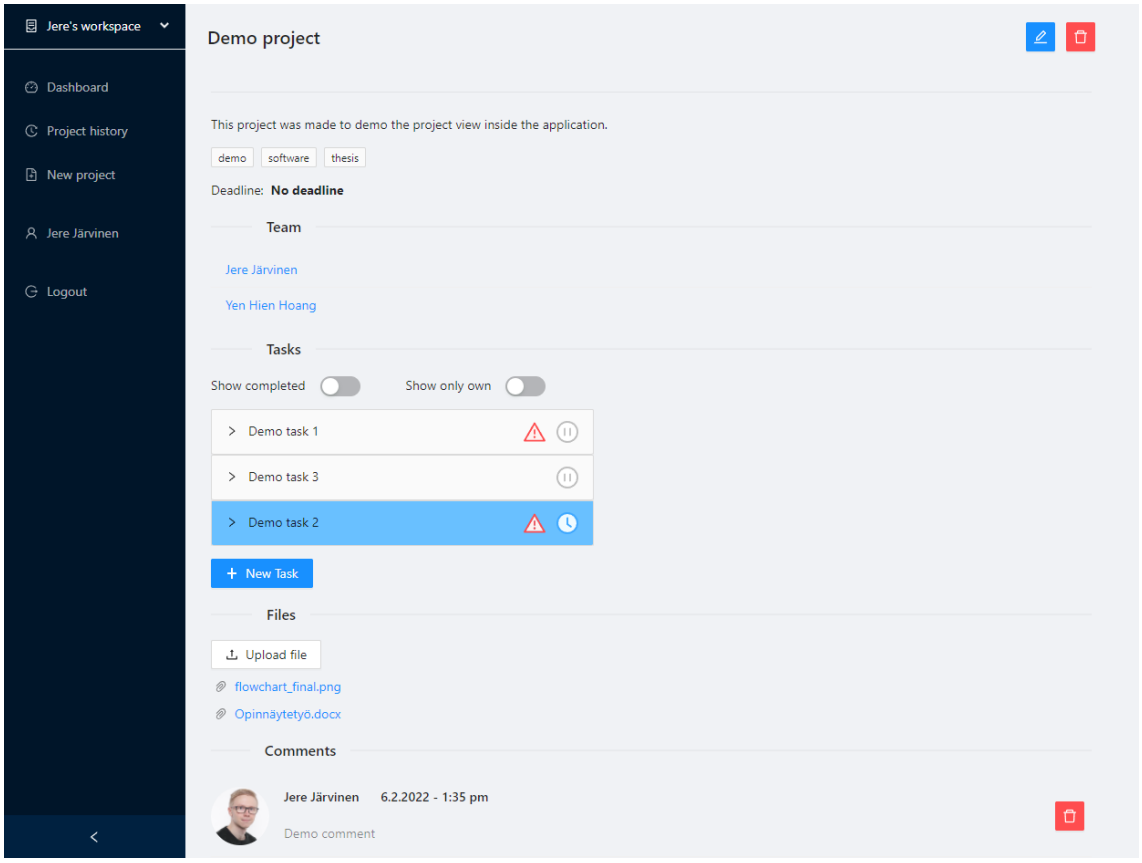
Kuva 7: Dasboard-sivu.

Laajemmat tiedot voidaan nähdä porautumalla itse projektin sivulle korttia klikkaamalla. Projektisivun sisältö muodostuu seuraavista osioista:

- Nimi
- Mahdollinen asiakas

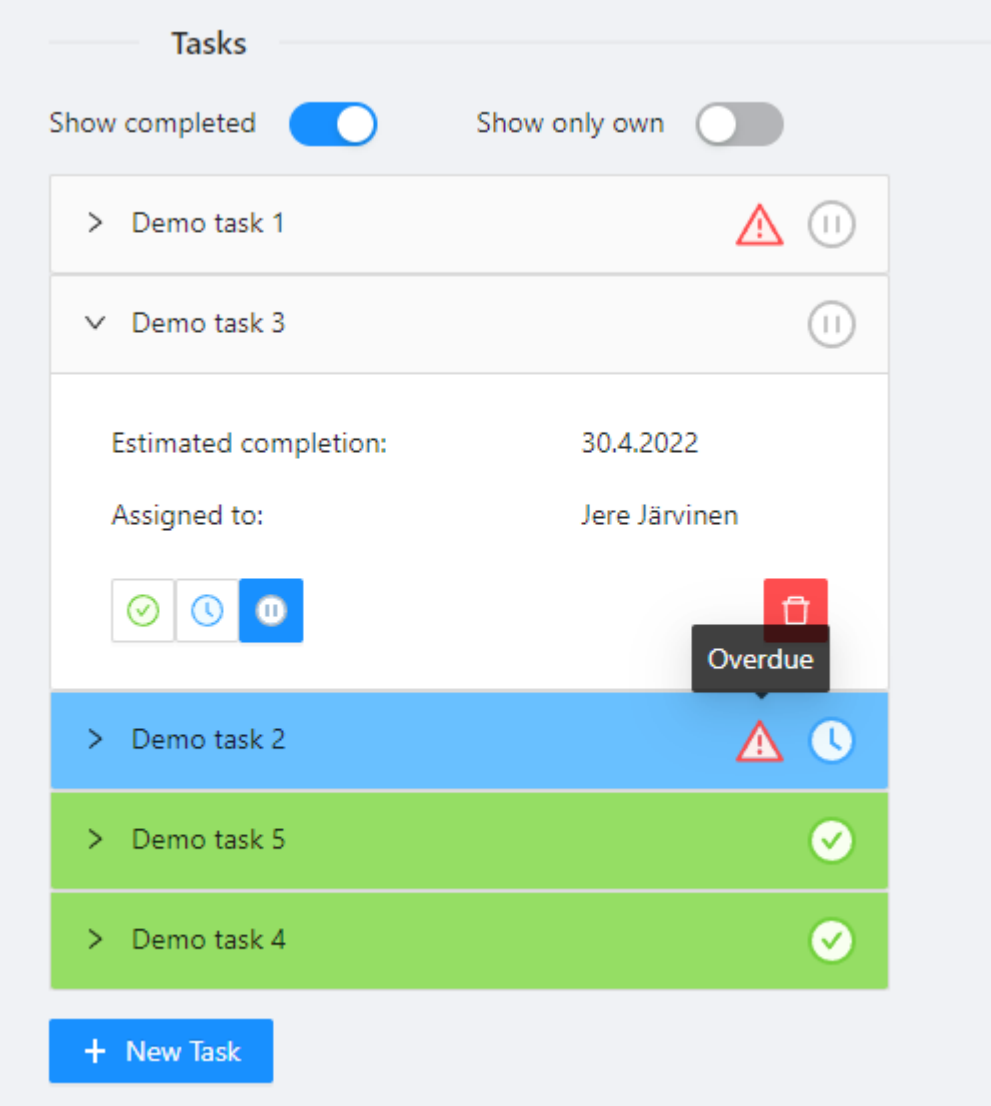
- Kuvaus
- Tunnisteet (tagit)
- Deadline
- Projektitiimi
- Liitetiedostot ja niiden lataaminen
- Kommentit
- Tehtävien hallinta

Lisäksi sivu sisältää toiminnot projektin tietojen muokkaamiseen sekä sen poistamiseen. Uuden projektin luominen tapahtuu erillisen sivun kautta, mutta projektin luominen on mahdollista vain, jos käyttäjällä on käytössään ainakin yksi työtila.



Kuva 8: Projektisivu.

Tehtävien hallinta on rakennettu projektisivun sisään. Tehtävät näytetään listana, jonka jäseniä voi laajentaa sisällön tarkastelemista ja toimintojen suorittamista varten. Tehtävää ei voi toistaiseksi muokata, mutta sen tilaa voidaan muuttaa (ei aloitettu / työn alla / valmis) ja se voidaan poistaa kokonaan. Oletuksena tehtävälistassa näytetään kaikkien projektitiimin jäsenten tehtävät, mutta valmistuneet tehtävät ovat piilotettuna. Näkyvyyksiä voidaan säätää kytkinvalinnoilla ”näytä valmistuneet” sekä ”näytä vain omat”.



Kuva 9: Tehtävät.

4.3 Ominaisuuksien jatkokehitys

Tämän opinnäytetyön puitteissa sovelluksesta tehtiin vähimmäisvaatimukset täyttävä, toimiva prototyyppi, jota voidaan hyödyntää sellaisenaan sekä kehittää edelleen tulevaisuudessa. Tässä kappaleessa käsittelen suunnitelmia sovelluksen jatkokehitykseen.

Merkittävin prototyypin ulkopuolelle jäänyt ominaisuus oli mahdollisuus tehtävien priorisoimiseen ja resursoimiseen. Nykyisessä versiossa tehtäville asetetaan arvioitu valmistumispäivä, mutta tätä tietoa ei hyödynnetä laajemmin. Valmistumispäivän lisäksi tehtäville olisi syytä asettaa laajuus esimerkiksi vapaamuotoisen pistemäärän tai aika-arvion (esim. tunteina / päivinä) muodossa. Tätä tietoa olisi mahdollista hyödyntää esimerkiksi uusien tehtävien jakamisessa tiimin jäsenten kesken. Tehtävien työmäärien kautta saadaan myös tietoa projektin kokonaistyömäärästä ja sitä kautta helpotusta tulevien samankaltaisten projektien resursointiin.

Tämän ominaisuuden myötä tehtävien hallintaa on syytä laajentaa mahdollistamalla tarkempi tekstimuotoinen kuvaus sekä tehtävän muokkaaminen, jotta työmääräarviota voidaan tarvittaessa korjata. Tehtävien muokattavuuden myötä voidaan myös mahdollistaa niiden luominen ilman ennalta asetettua vastuuhenkilöä tai arvioitua valmistumisaikaa. Projektin tehtävät ovat usein toisistaan riippuvaisia, esimerkiksi siten että tiettyä tehtävää ei voida aloittaa tai suorittaa loppuun ennen jonkin toisen tehtävän valmistumista. Näiden riippuvuuksien kuvaaminen on myös olennainen jatkokehitysidea sovellukselle.

Projektien budjetoiminen ja sen täyttymisen seuraaminen jäi tässä vaiheessa sovelluksen skaalan ulkopuolelle. Tämä on kuitenkin yksi potentiaalisimmista uusista ominaisuuksista jatkokehitystä ajatellen. Vaikka budjetointi ei ole relevantti ominaisuus koko sovelluksen kohderyhmälle, on se tärkeä erityisesti kaupallisten toimijoiden näkökulmasta. Näin ollen budjetointi olisi syytä tuoda valittavaksi työtilakohtaisesti asetusten kautta.

Jatkossa sovellusta voidaan laajentaa tuomalla mahdollisuus asiakasorganisaatioiden ja niiden yhteyshenkilöiden tallentamiseen. Yhteyshenkilöt ja niiden yhteystiedot ovat projektin kannalta erittäin relevanttia tietoa, joten on perusteltua saada ne helposti nähtäville suoraan projektisivun kautta. Käytännössä asiakkaiden ja yhteyshenkilöiden listaamiseen sekä luomiseen voidaan tehdä erilliset sivut. Yksittäiset asiakasorganisaatiot ja yhteyshenkilöt olisivat avattavissa omiin näkymiinsä tarkempaa tarkastelua tai muokkausta varten ja projektisivuilta pääsisi näihin näkymiin suorien linkkien kautta, kuten projektitiimin jäsenten kohdalla (ks. kuva 5).

Pidemmille tekstikentille, kuten projektin kuvaukselle sekä kommenteille on toteutettava ns. rich text -kentät, jotka sallivat muotoilujen käyttämisen. Nykyisellään kyseiset tekstisyötteet ottavat vain merkkijonoja sellaisenaan, mutta tuomalla mahdolliseksi HTML:n sisältämät muotoiluvaihtoehdot, saadaan teksteistä paremmin jäsennettyjä ja miellyttävämpiä lukea.

Dashboard-näkymää on mahdollista kehittää eteenpäin lisäämällä sen muokattavuutta. Esimerkkeinä käyttäjälle voidaan tulevaisuudessa tuoda mahdollisuudet valita oletusarvoisen korttinäkymän sekä taulukkomuotoisen näkymän välillä. Mikäli käyttäjällä on aktiivisessa tarkastelussa suuri määrä projekteja, voidaan niitä tuoda taulukkomuodossa enemmän yhteen näkymään. Lisäksi suunnitelmissa on kytkinvalinta, jolla voidaan valita näytettäväksi vain projektit, joissa käyttäjä on itse osallisena. Tämä on hyödyllinen ominaisuus mm. tilanteissa, joissa samaa työtilaa käyttää useampi tiimi.

Nykyisessä prototyypissä sovelluksen responsiivisuuteen ei ole kiinnitetty juurikaan huomiota. Vaikka palvelu on tarkoitettu ensisijaisesti tietokoneella käytettäväksi, on sen jonkinasteinen toimivuus varmistettava myös mobiililaitteilla. Olemassa olevaa React-käyttöliittymää on myös mahdollista hyödyntää jatkossa esim. React Nativella toteutetun mobiilisovelluksen kehittämisessä.

5 Toteutus

5.1 Tietokanta

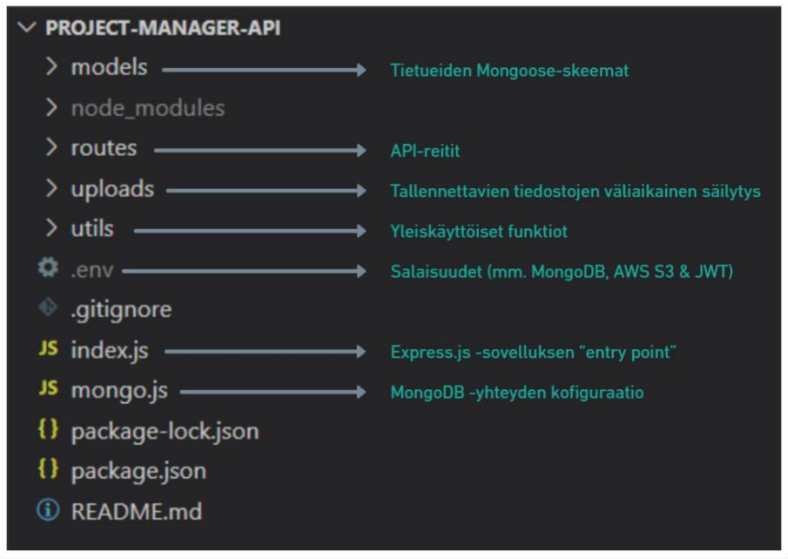
Sovelluksen tietokantaratkaisuna toimii MongoDB. Sovelluksen tietorakenteen ollessa verrattain kevyt, oli perusteltua valita NoSQL-tyyppinen tietokanta, joiden perusajatuksena on kyselyjen nopeuden optimoiminen tiedon säilytystilan kustannuksella. Tietueiden välisiä riippuvuuksia, eli relaatioita on sovelluksessa vain vähän, joten siinäkään suhteessa relaatiotietokannan (SQL) vahvuudet eivät pääsisi kyseisessä sovelluksessa oikeuksiinsa. MongoDB:n puolesta puhuivat lisäksi sen helppokäyttöisyys Node.js -pohjaisessa palvelinratkaisussa, sekä ilmainen tallennustila prototyyppiä varten.

Tietokannan rakenne noudattelee aikaisemmin suunniteltua kaaviota sovelluksen rakenteesta. Näin ollen se rakentuu neljästä dokumenttikokoelmasta (collection), jotka ovat nimeltään käyttäjä, työtila, projekti ja tehtävä. Sovelluksen back-endissä tietokannan skeemojen mallintamiseen datan validoimiseen sekä tietokantakyselyihin käytetään Mongoose-kirjastoa.

Poikkeuksena muusta datasta, staattiset tiedostot esim. käyttäjien profiilikuvat tai projektien liitetiedostot säilötään Amazon Web Servicesin (AWS) S3-palvelun avulla. S3 on luonteeltaan yksinkertainen pilvitallennuspalvelu, jonka käyttöön on olemassa kehittäjäystävälliset työkalut useille ohjelmointikielille/alustoille, mukaan lukien tässä sovelluksessa käytetty Node.js.

5.2 Back-end

Sovelluksen back-endinä toimii Node.js -pohjainen palvelinsovellus, joka toimii rajapintana tietokannan ja React-sovelluksen välillä. Rajapinta ottaa asiakkaalta vastaan pyyntöjä, jotka voivat olla luonteeltaan datan hakemista tai erilaisten toimintojen suorittamista dataan liittyen. Esimerkkejä rajapintapyynnöistä ovat vaikkapa käyttäjän työtilojen hakeminen, projektin nimen muuttaminen tai tehtävän poistaminen. Rajapinta on rakennettu kokonaan Express.js -kirjaston avulla.



Kuva 10: Back-endin rakenne.

Express.js -sovelluksen rakenne on hyvin yksinkertainen. Mongoose-kirjaston käyttämät tietokannan dokumenttiskeemat sijaitsevat omissa tiedostoissaan models-kansiossa. Skeemoissa on määritelty dokumenttien sisältämien kenttien tietotyypit sekä muita validaatioita, esim. arvon pituus, kentän pakollisuus tai ainutlaatuisuus.

Routes-kansioon on eritelty kaikki rajapinnan tarjoamat reitit skeemoja vastaavalla tiedostorakenteella. Esimerkiksi:

- **Skeema:** *models/Project.js*
- **Skeemaan liittyvät reitit:** *routes/projects.js*

Jokainen routes-kansion tiedosto määrittelee erillisen Express-reitittimen, jota käytetään välifunktiona itse Express-sovelluksen määrittelemässä Index.js -tiedostossa (ks. kuvio 7). Käyttäjän tunnistautumiseen (kirjautuminen/rekisteröityminen) liittyvät reitit ovat erillisen reitittimen (authRouter) vastuulla, sillä niiden yhteydessä käyttäjältä ei vielä vaadita voimassa olevaa tunnistautumista. Varsinaista dataa käsittelevissä reiteissä tarkistetaan, että pyynnön tekijällä on tarvittavat oikeudet toiminnon suorittamiseen.

```
const app = express();

app.use(express.json());
app.use(cors());

app.use("/auth", authRouter);
app.use("/workspaces", authenticator, workspacesRouter);
app.use("/projects", authenticator, projectsRouter);
app.use("/users", authenticator, usersRouter);
app.use(errorHandler);
```

Kuva 11: Index.js – välifunktiot

Yllä kuvatuista välifunktioista ensimmäinen, express.json, on expressin sisään rakennettu vakio-välifunktio, joka vastaa sisään tulevien pyyntöjen JSON-sisältöjen automaattisesta parsimisesta. Cors-välifunktio mahdollistaa pyyntöjen tekemisen muilta palvelimilta, esimerkiksi paikallisen laitteen eri portissa ajettavasta frontend-sovelluksesta. Tuotannossa kyseinen välifunktio vaatisi tarkempia asetuksia, mutta kehitysvaiheessa välifunktio on sellaisenaan riittävä, sallien liikenteen kaikista lähteistä kaikkiin reitteihin. Seuraavat välifunktiot ovat aikaisemmin mainittuja reitittimiä. Lopuksi sovellus kutsuu tarvittaessa errorHandler-välifunktiota, joka sisältää lähes kaiken virheidenkäsittelylogiikan. Sen kautta pyritään saamaan asiakkaalle mahdollisimman tarkka kuvaus tapahtuneista virheistä.

5.2.1 Käyttäjän tunnistautuminen

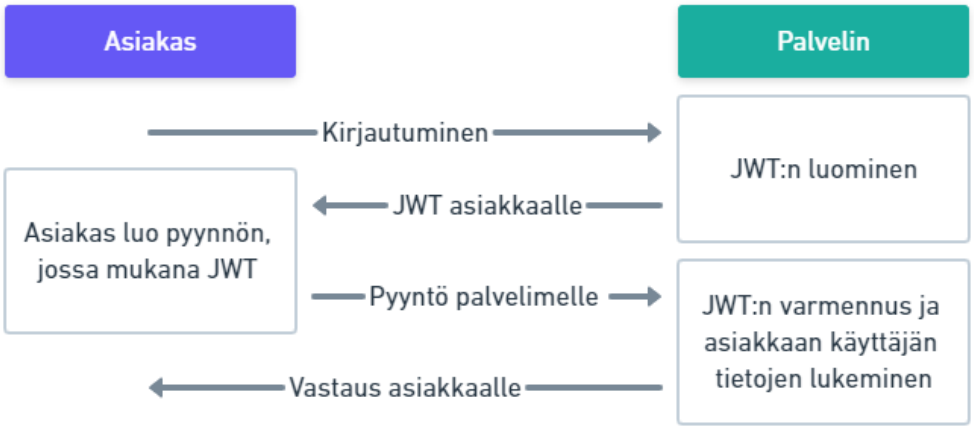
Jotta sovelluksen käsittelemät tiedot olisivat saatavilla ja muokattavissa vain käyttäjille, joilla siihen on oikeus, täytyy palvelimelle tulevan pyynnön yhteydessä tarkistaa sen lähettäjän identiteetti.

Käyttäjän tunnistautuminen tapahtuu tämän työn sovelluksessa merkkijonopohjaisella tunnistautumisella JSON Web Tokenien (JWT) avulla. JWT on standardisoitu (ehdotettu RFC-standardi 7519) tapa välittää tietoa turvallisesti ja luotettavasti JavaScript -objektien muodossa. JWT koostuu kolmesta osiosta, jotka ovat otsikko (header), sisältö (payload) ja allekirjoitus (signature). Otsikko sisältää tyypillisesti tiedot tunnistautumiseen käytettävän menetelmän tyypistä sekä allekirjoituksessa käytetystä salausalgoritmista. Sisältö-osio sisältää tietoja käyttäjästä ja JWT:stä itsestään, esimerkkeinä käyttäjän ID, käyttöoikeudet sekä JWT:n voimassaolo. (jwt.io, 2022)

JWT liikkuu verkon yli salatussa (enkoodatussa) muodossa merkkijonona, jossa sen kolmea osiota vastaavat osuudet on eroteltu pisteillä (xxxx.yyyy.zzzz). JWT:n allekirjoitus

muodostetaan sen salauksen yhteydessä otsikon, sisällön sekä yksityisen salausavaimen avulla. JWT:n ”allekirjoittamisella” salaista avainta käyttämällä voidaan varmistaa, että sen sisältö ei ole muuttunut matkan varrella ja että sen lähettäjä on siinä määritelty taho. (jwt.io, 2022)

Onnistuneen sisäänkirjautumisen yhteydessä asiakkaalle annetaan allekirjoitettu token, joka säilötään asiakkaan laitteella paikallisesti. JWT:lle on suositeltavaa asettaa vanhenemisaika, jonka jälkeen sillä ei voida enää tehdä pyyntöjä palvelimelle. Vanhenemisen tarkoituksena on minimoida haitta, mikäli JWT päätyy vääriin käsiin.



Kuvio 7: JWT:n toiminta.

Sovelluksen koodissa JWT:n tunnistaminen tapahtuu tunnistautumis-välifunktiossa (authorization). Uuden käyttäjän luomista ja sisäänkirjautumista lukuun ottamatta kaikki muut API-reitit kulkevat ensin kyseisen välifunktion läpi. Välifunktio poimii pyynnöstä authorization-nimisen http-otsikon (header), jossa on vakiintuneen käytänteen mukaisesti etuliitteenä sana ”Bearer”. Tunnistautumiseen käytettävä arvo erotetaan otsikon merkkijonosta ja sen validiteetti tarkistetaan JWT:n verify-funktion avulla. Kuten aiemmin mainittua, tarkistusfunktio vaatii siis yksityisen salausavaimen ja kyseinen avain säilötään tietoturvasyistä ns. ympäristömuuttujassa. Tällä tarkoitetaan muuttujia, joita ei ole määritelty lähdekoodissa, vaan ainoastaan ohjelman suorittavalla laitteella. Näin ollen niitä ei varastoida esim. Githubissa. Jos token verifioidaan onnistuneesti, siirtyy pyynnön käsittely eteenpäin. Mikäli tunnistautumismerkkijonoa ei löydy tai se ole validi, palautetaan rajapinnasta virhe ja käyttöliittymässä käyttäjä siirretään kirjautumissivulle.

```
const jwt = require("jsonwebtoken");
require("dotenv").config();

module.exports = (req, res, next) => {
    const authHeader = req.headers["authorization"];
    const accessToken = authHeader && authHeader.split(" ")[1];

    if (accessToken == null) return res.status(401).send({ messages: "No access token" });

    jwt.verify(accessToken, process.env.JWT_SECRET, (err, user) => {
        if (err) return res.status(401).send({ messages: "Invalid access token" });
        req.user = user;
        next();
    });
};
```

Kuva 12: Käyttäjän tunnistautumiseen käytettävä välifunktio Express-sovelluksessa.

5.3 Käyttöliittymä

Käyttöliittymien kehittäminen Reactilla perustuu monikäyttöisten komponenttien hyödyntämiseen. Kehittäjän on mahdollista luoda kaikki käyttämänsä komponentit itse, mutta Reactin ympärille syntynyt ekosysteemi tarjoaa suuren määrän valmiita käyttöliittymäkirjastoja, jotka sisältävät pitkälle jalostettuja komponentteja tavallisimmille käyttöliittymätoiminnoille. Etuna kyseisten kirjastojen käyttämisessä on yhtenäisen ulkoasun helppo ylläpitäminen läpi koko sovelluksen. Lisäksi kehitystyö nopeutuu huomattavasti, kun ei ole tarvetta luoda sovelluksen peruskomponentteja joka projektin yhteydessä uudestaan. Tässä projektissa hyödynnettiin Ant Design -nimistä käyttöliittymäkirjastoa, joka on yksi alan laajimmista ratkaisuista.

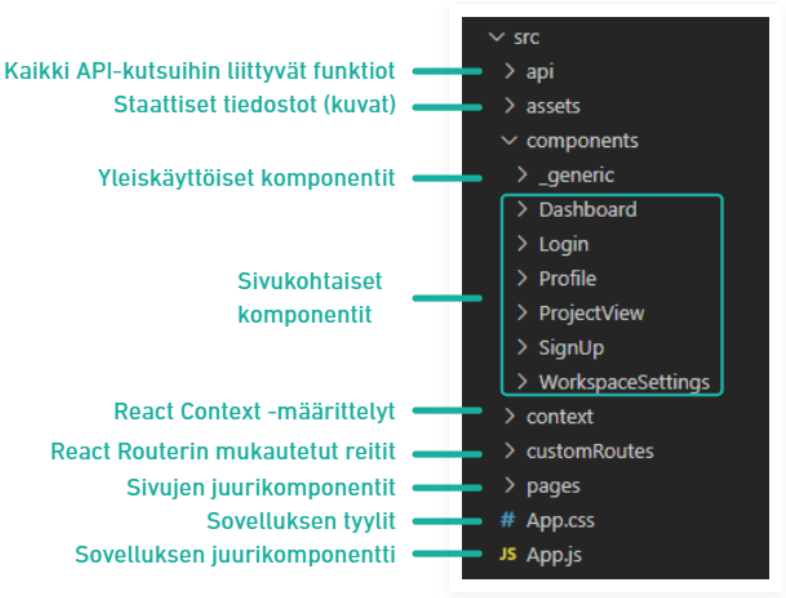
Tämän projektin React-sovellus on tyypiltään ns. Single Page Application (SPA), jossa erillisten tiedostojen välillä navigoimisen sijaan sovellus käyttää yhtä HTML-tiedostoa, jonka sisältöä päivitetään dynaamisesti tarpeen mukaan ilman sivun uudelleenlatausta. (developer.mozilla.org, 2022b)

Sovelluksen pohjana olevan tiedoston pysyessä samana, ei siihen lähtökohtaisesti liity URL-reititystä. Käytännössä sivujen yksilöiminen URL:n perusteella on kuitenkin tarpeen navigoinnin ja linkittämisen mahdollistamiseksi. React ei itsessään sisällä reititykseen liittyviä ominaisuuksia, mutta tähän tarkoitukseen on olemassa useita JavaScript-kirjastoja, jotka mahdollistavat komponenttien päivittämisen URL:n perusteella ilman sivun lataamista uudelleen. Tämän työn sovelluksessa reititykseen käytetään React Router -nimistä kirjastoa. (Ganatra, 2018)

Sovelluksen reiteistä vain login ja sign-up ovat käytettävissä ilman tunnistautumista. Muut reitit ovat ”yksityisiä”, eli niiden kohdalla tarkistetaan, onko sovellukselle asetettu aktiivinen käyttäjä. Mikäli käyttäjän tietoja ei ole, ohjaa yksityinen reitti kirjautumissivulle. Kirjautunut käyttäjä säilytetään sovelluksessa React Contextin avulla. Context on yksinkertainen tapa

säilyttää tietoa siten, että se on kaikkien komponenttien käytettävissä. Käyttäjän tiedot säilötään lisäksi localStorageen, eli väliaikaisesti käyttäjän laitteelle. Näin ollen uutta sisäänkirjautumista ei tarvita edes selainikkunan sulkemisen jälkeen, mikäli tunnistautuminen on vielä voimassa.

Sovelluksen sivujen perusasettelu muodostuu sivupalkkinavigaatiosta sekä sivun varsinaisesta sisällöstä. Tämä asettelu on toteutettu PageLayout-komponentilla, jonka sisään yksittäisen sivun sisällöstä vastaavat komponentit sijoitetaan. Sovelluksen tiedostorakenteessa sivujen juurikomponentit sijaitsevat pages-kansiossa. Kaikki alemman tason komponentit, joista sivut muodostuvat, sijaitsevat components-kansiossa, joka on edelleen jaoteltu kansioihin sen mukaan, millä sivulla kyseinen komponentti on käytössä. Geneeriset komponentit löytyvät omasta kansiostaan. API-kutsuja tekevät funktiot on eritelty omiin tietuetyyppikohtaisiin moduuleihinsa, jotka sijaitsevat api-kansion sisällä.



Kuva 13: React-sovelluksen kansiorakenne.

5.4 Toteutuksen jatkokehitys

Sovelluksen toteutuksen edetessä kävi ilmeiseksi, että puhtaan JavaScriptin käyttämisellä ohjelmointikielenä on jonkin verran haittapuolia. Vahvan tyypityksen puute ja koodin tulkkaaminen ajon aikana aiheuttavat JavaScript-kehityksessä herkästi huomaamatta jääneitä bugeja sekä hankaluukisa bugien ratkomisessa.

Suosittu ratkaisu kyseisiin ongelmiin on JavaScriptin syntaksia laajentava ohjelmointikieli TypeScript, joka mahdollistaa vahvan tyypityksen. TypeScript koodi vaatii käännösvaiheen, jossa se muunnetaan puhtaaksi JavaScriptiksi. Käännösvaihe yhdistettynä vahvaan tyypitykseen

auttaa saamaan kiinni bugeja, jotka puhtaalla JavaScriptillä olisivat olleet havaittavissa vasta koodia suoritettaessa. Lisäksi TypeScript tarjoaa parempaa käytettävyyttä koodieditoreissa mm. automaattisten ehdotusten muodossa. (Fireship, 2022)

TypeScript on nopeasti yleistyvä ja pidetty teknologia modernissa JavaScript-kehityksessä ja mikäli sovellusta lähdetään jatkokehittämään suuremmassa mittakaavassa, on sen käyttöönotto yksi tärkeimmistä askelista.

Sovelluksen käyttökokemuksen parantamiseksi on käyttäjän tunnistamisen yhteydessä syytä käyttää ns. päivitystunnistetta (refresh token). Sovellus käyttää päivitystunnistetta uuden voimassa olevan JWT:n pyytämiseen palvelimelta automaattisesti vanhan vanhentuessa ja sillä ei ole mahdollista tehdä muita API-pyyntöjä. Päivitystunniste on yleensä voimassa huomattavasti pidempään kuin varsinainen API-kutsujen tekemiseen käytettävä tunniste. Näin varsinaisen tunnisteen voimassaoloaika voidaan pitää lyhyenä, mutta asiakkaan ei tarvitse kirjautua uudestaan sisään aina sen vanhetessa. Päivitystunnisteen toteutus on yksi sovelluksen tärkeimmistä jatkokehityskohteista.

6 Yhteenveto

Tämän opinnäytetyön lopputuloksena syntyi pienten projektien hallintaan käytettävä verkkosovellus. Sovellukselle on kartoitettu lukuisia jatkokehityskohteita, mutta tämän työn kontekstissa tarkoitus oli saada aikaan ydinominaisuuksiltaan minimivaatimukset täyttävä prototyyppi ja se tavoite täyttyi. Sovellus on sellaisenaan valmis hyödynnettäväksi ja julkaistavaksi. Lopputuloksena syntynyttä ohjelmistoa käytettiin myös työkaluna itse opinnäytetyöprojektin hallintaan erityisesti tekniseen toteutukseen liittyvien tehtävien puitteissa.

Sovelluksen suunnittelun ja kehityksen taustana käsiteltiin projektinhallinnan yleistä teoriaa, sen menetelmiä, ohjelmistojen roolia sekä ominaispiirteitä pienten projektien kontekstissa. Sovelluksen kehityksessä käytettäviin teknologioihin otettiin myös katsaus teoriatasolla. Itse sovelluksen toiminta ja sisältö käytiin läpi omassa kappaleessaan ja lopuksi syvennyttiin tekniseen toteutukseen ja siinä tehtyihin ratkaisuihin.

Lähteet

Painetut

Boduch, A., & Derks, R. (2020). React and react native: A complete hands-on guide to modern web and mobile development with react. js, 3rd edition.

Dinsmore, P. C. & Cabanis-Brewin, J. (2014). The ama handbook of project management.

Fowler, A. (2015). Nosql for dummies.

Ganatra, S. (2018). React router quick start guide: Routing in react applications made easy. Packt Publishing, Limited.

Hoque, S. (2020). Full-stack react projects: Learn mern stack development by building modern web apps using mongodb, express, react, and node. js, 2nd edition.

Kerzner, H. & Kerzner, H. R. (2013). Project management: A systems approach to planning, scheduling, and controlling. John Wiley & Sons, Incorporated.

Mardan, A. (2018). Practical node.js: Building real-world scalable web apps. Apress L. P.

Olson, D. L. (2020). Core concepts of project management.

Oluyege, P. (2019). Mongodb, express, angular, and node.js fundamentals: Become a mean master and rule the world of web applications.

Phillips, J. (2011). Project management for small business: A streamlined approach from planning to completion.

Portny, S. E. (2013). Project management for dummies. John Wiley & Sons, Incorporated.

Singhal, M. & Corvalan, D. (2016). Getting started with react. Packt Publishing, Limited.

Wells, K. N. & Kloppenborg, T. J. (2018). Project management essentials, second edition.

Westcott, R. (2004). Simplified project management for the quality professional: How to plan for and manage small and medium-size projects. ASQ Quality Press.

Sähköiset

Codeacademy.com. (11.2.2022). React: The Virtual DOM. https://www.codecademy.com/article/react-virtual-dom

Developer.mozilla.org. (29.1.2022a). What is JavaScript? https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

Developer.mozilla.org. (10.3.2022b). SPA (Single-page application). https://developer.mozilla.org/en-US/docs/Glossary/SPA

Expressjs.com. (28.1.2022a). Using middleware. https://expressjs.com/en/guide/using-middleware.html

Expressjs.com. (28.1.2022b). Routing. https://expressjs.com/en/guide/routing.html

Fireship. (20.2.2022). TypeScript in 100 Seconds. https://www.youtube.com/watch?v=zQnBQ4tB3ZA&ab_channel=Fireship

Geeksforgeeks.org. (6.12.2021). MongoDB – Database, Collection, and Document. https://www.geeksforgeeks.org/mongodb-database-collection-and-document/

Jwt.io. (3.2.2022). Introduction to JSON Web Tokens. https://jwt.io/introduction

Kundu, J., Bishoi, T., Bhattacharya, M. & Chowdhury, A. (2015). Viitattu 14.3.2022. PROJECT MANAGEMENT SOFTWARE - AN OVERVIEW. https://www.academia.edu/17776608/PROJECT_MANAGEMENT_SOFTWARE_AN_OVERVIEW

MongoDB. (28.9.2021a). MERN Stack Explained. https://www.mongodb.com/mern-stack

MongoDB. (28.9.2021b). What is NoSQL? https://www.mongodb.com/nosql-explained

MongoDB. (6.1.2022a). MongoDB & Mongoose: Compatibility and Comparison. https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/

MongoDB. (22.1.2022b). Database as a Service (DBaaS) Explained. https://www.mongodb.com/database-as-a-service

Reactjs.org. (18.2.2022). JSX In Depth. https://reactjs.org/docs/jsx-in-depth.html

Rowe, S. F. (2007). Viitattu 10.4.2022. Managing and leading small projects. Paper presented at PMI® Global Congress 2007—North America, Atlanta, GA. Newtown Square, PA: Project Management Institute. https://www.pmi.org/learning/library/managing-leading-small-projects-7245

Web Dev Simplified. (3.2.2022). What Is JWT and Why Should You Use JWT.
https://www.youtube.com/watch?v=7Q17ubqLfaM&t=3s&ab_channel=WebDevSimplified

Kuviot

Kuvat