Kimmo Perälä

Web-pohjainen lomasuunnittelusovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

5.5.2022

Tiivistelmä

Tekijä:	Kimmo Perälä
Otsikko:	Web-pohjainen lomasuunnittelusovellus
Sivumäärä:	45 sivua
Aika:	5.5.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Juha Kämäri

Insinöörityössä määriteltiin, suunniteltiin ja toteutettiin alustava versio web-sovelluksesta lomasuunnitteluun tilaajayrityksen (Tieto Finland Oy) työryhmän sisäiseen käyttöön. Lomasuunnittelusovellus on tarkoitettu työryhmän työntekijöiden loma-aikojen suunnittelua, ilmoittamista, tarkastelua ja yhteensovittamista varten. Sovellusta varten vertailtiin ja valittiin eri vaihtoehdoista teknologiapino, ohjelmointirajapintaprotokolla ja käytettävät komponentit.

Teknologiapinoksi vertailtiin muutamia nykyään suosittuja vaihtoehtoja, joita ovat esimerkiksi MEAN ja LAMP. Teknologiapinoksi valittu MERN-pino koostuu MongoDB-tietokannasta, Express-web-kehyksestä, React-kirjastosta ja Node.js-suoritusympäristöstä. Ohjelmointirajapintaprotokollista valittiin REST-rajapinta. Sovellukseen suunniteltiin myös tuleva reaaliaikaisten ilmoitusten ratkaisu Slack-viestintäsovellukseen.

Insinöörityössä kuvataan vertaillut ja valitut teknologiat sekä esitellään myös yleistä sovelluksen määrittelyä, suunnittelua ja alustavaa toteutusta. Siinä käsitellään myös ajan määrittelemistä JavaScriptissa ja automaattisten Slack-viestien luomista. Sovelluksen rakenne koostuu lomien CRUD-lomakkeesta, lomakkeen kalenteriponnahdusikkunasta, yleisnäkymästä ja ylläpitosivusta. Lomasuunnittelusovelluksen alustavaa versiota esitellään käyttöliittymäkuvilla, koodiesimerkeillä ja lokeilla.

Teknologiapinon valinta tuntui oikealta, ja useita sovelluksen määrittelyn tavoitteita saavutettiin jo toteutetussa alustavassa versiossa. Jatkossa sovellus tulisi siirtää pilvipalveluun, lisätä suunniteltu Slack-integraatio ja tehdä käyttäjätestausta työryhmässä.

Avainsanat: MERN, web-kehitys, full stack, API, React

Abstract

Author:	Kimmo Perälä
Title:	Web-based holiday planning application
Number of Pages:	45 pages
Date:	5 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Juha Kämäri, Senior Lecturer

The thesis covers defining, planning and developing the initial version of a holiday planning web application for internal use of the commissioner company's (Tieto Finland Ltd) team. The holiday planning application serves as a planning, reporting, monitoring and coordinating tool for staff holidays. Multiple options were compared to choose the software stack, application programming interface protocol and web components.

Web stacks currently widely used, such as MEAN and LAMP were considered for the application. The chosen software stack MERN consists of MongoDB database, Express web framework, React library and Node.js web server. REST API was chosen as the application programming interface protocol. An upcoming Slack push notification solution was also planned for the application.

The thesis describes the compared and chosen technologies and introduces the defining, planning and development of the initial version of the application. It also presents time handling in JavaScript and creating push notifications for Slack. The structure of the application consists of a vacation CRUD-form, a calendar popup of the form, a table view and an admin page. The initial version of the holiday planning application is presented with user interface pictures, code examples and logs.

The chosen web stack achieved the application definition goals for the initial version of the holiday planning application. In the future the application will be migrated to a cloud service with planned Slack integration and user testing in the team will be performed.

Keywords: MERN, web development, full stack, API, React

Sisällys

Lyhenteet ja käsitteet

API: *Application programming interface*. Ohjelmointirajapinta. Tekninen määrittely, joka kuvaa tiedonsiirtovaihtoehtoja ratkaisujen välillä sekä tämän määrittelyn mukainen ohjelmisto.

Azure: Microsoftin julkinen pilvipalvelu.

CRUD: *Create, read, update, delete*. Tiedontallennuksen perustoiminnot: datan luominen, lukeminen, muokkaaminen sekä poisto.

DOM: *Document Object Model*. HTML-dokumentin kuvaus puuna, jonka solmut ovat olioita, jotka edustavat tiettyä dokumentin osaa ja joita voidaan hakea, tarkastella ja muokata esimerkiksi JavaScriptillä.

Fullstack: Ohjelmistoteknologioiden kokonaisuus, jolla ohjelmistoratkaisu on toteutettu ja joka koostuu asiakaspuolesta (frontend) ja palvelinpuolesta (backend).

HTTP: *Hypertext Transfer Protocol*. Selainten ja palvelinten käyttämä tiedonsiirtoprotokolla, joka siirtää web-sivun sisällön palvelimelta asiakkaalle.

JSON: *JavaScript Object Notation*. Ihmisen luettava, ohjelmointikielistä riippumaton ja tekstimuotoinen datan tallentamis- ja siirtotiedostomuoto, jota käytetään etenkin verkkosovelluksissa.

MERN: Fullstack-ratkaisu, joka koostuu MongoDB-tietokannasta, React-JavaScript-kirjastosta, Node.js-suoritusympäristöstä ja sen Express.js-kehyksestä.

NoSQL: *Not only SQL*. Relaatiomallista (joka perustuu taulukoiden riveihin ja sarakkeisiin sekä yhdistäviin tunnisteisiin) poikkeava tietokantatyyppi, joka tukee joustavia datamalleja ja skaalautuu.

Protokolla: Sääntöjärjestelmä, joka määrittelee laitteiden tai ohjelmien väliset yhteydet ja tiedonsiirron.

Slack: Työviestintäsovellus, jossa kommunikoidaan pääosin suorilla viesteillä ja kanavien ryhmäkeskusteluissa. Kanaviin voi lähettää myös automaattisia viestejä toisista järjestelmistä.

URI: *Uniform Resource Identifier*. Sijainnin, nimen tai molemmat sisältävä tietoverkkotunniste. Sijainnin sisältävä URI on URL (Uniform Resource Locator), jota käytetään web-sivujen osoittamiseen.

UTC: *Coordinated Universal Time*. Koordinoitu yleisaika on aikajärjestelmä, jolla määritellään maailman aika. Se pysyy sekunnin sisällä nollapituuspiirin aurinkoajasta.

1 Johdanto

Insinöörityössä määritellään, suunnitellaan ja toteutetaan tilaajayrityksen (Tieto Finland Oy) lomasuunnittelusovellus (myöhemmin lomasovellus) yrityksen työryhmän sisäiseen käyttöön. Tärkeänä osana suunnittelua valitaan sovellukselle teknologiapino (stack), ohjelmointirajapintaprotokolla ja käytetyt komponentit. Teknologiapinoista tehdään kattava vertailu, jonka perusteella valitaan sovellukselle sopiva pino. Tieto Finland Oy on osa Tietoevry Oyj:tä, jossa on yli 24 000 työntekijää maailmanlaajuisesti ja jonka vuotuinen liikevaihto on noin 3 miljardia [1]. Insinöörityö tehdään yrityksen Create-toimialueen Customer experience -yksikköön jatkuvien palveluiden tiimiä varten. Tiimissä on jo pidemmän aikaa ollut tarvetta verkkosovellukselle, jolla voitaisiin hallinnoida työntekijöiden lomien suunnittelua. Sovelluksella voisi olla käyttöä laajemminkin yrityksessä.

Lomasovelluksen käyttötarkoituksena on, että työryhmän työntekijät kirjaavat siihen suunnittelemiensa lomien ajankohdat ja voivat tarkastella lomien yleistä, anonyymiä jakaantumista työryhmässä. Tällä hetkellä lomia varten yrityksessä on käytetty Microsoft Excel -taulukoita, joiden avulla työntekijät voivat suunnitella loma-aikojaan ja työnjohdossa seurataan, että työryhmässä on aina riittävästi työntekijöitä paikalla ja poissaolijoilla on valittuna sijainen. Jatkossa sovellus voisi tehdä seurannasta helpompaa ja automatisoidumpaa.

Insinöörityön tavoitteena on lomasovelluksen alustavan version määrittely, suunnittelu ja toteutus. Suunnitteluvaiheeseen kuuluvat tietenkin teknologiavalinnat, joita sovelluksen määrittely ohjaa. Sovellus tarvitsee web-teknologiapinon, ohjelmointirajapinnan ja valmiita avoimen lähdekoodin komponentteja sekä integroimisen Slack-viestintäsovellukseen. Tulevaisuudessa valmis sovellus siirretään Azure-pilvipalveluun ja tavoitteena on, että jatkokehitetty sovellus saataisiin työryhmän aktiiviseen käyttöön.

Insinöörityön pääaihe on tilaajayrityksen tarvitseman sovelluksen web-teknologiapinon valinta. Ohjelmistokehittäjä joutuu uransa aikana opettelemaan ja käyttämään eri teknologiapinoja, joten vertailu antaa hyvää tietoa myös tulevaisuuden ohjelmistotuotantoprosesseja varten. Insinöörityö kokoaa tietoa myös muista ajankohtaisista aiheista, kuten ohjelmointirajapinnoista ja valitulla teknologiapinolla ohjelmoinnista.

2 Lomasovelluksen määrittely

2.1 Lähtökohta

Tilaajayrityksen tiimissä on tarvetta lomasovellukselle, jossa työntekijät voisivat suunnitella ja merkata, kuinka jakaa vuosilomapäivänsä eri ajankohdille. Lomasovelluksen tarkoituksena on auttaa lomien yhteensovittamisessa ja alustavassa ilmoittamisessa sekä lomatilanteen tarkastelussa. Se ei toimisi virallisena lomien ja poissaolojen ilmoituspaikkana esimerkiksi palkkahallinnon käyttöön, vaan tähän yrityksellä on oma sisäinen ohjelmansa. Lomasovelluksen avulla voidaan reagoida töissä olevan henkilöstön määrän muutoksiin ja estää tilanteet, joissa lomalla olijoiden määrä olisi liian suuri tai lomailevalle työntekijälle ei löytyisi sijaista.

Tällä hetkellä lomien hallintatyökaluna ovat tiimin yhteiskäytössä olevat Microsoft Excel -taulukot, joihin loma-ajankohdat kirjataan päivämääräsarakkeisiin työntekijäkohtaiselle riville. Excel-taulukot ovat toimineet noin 20 henkilön tiimissä melko hyvin, mutta siihen liittyy myös muutamia ongelmia. Excel-taulukoista on hankala seurata yksittäisen työntekijän lomien käyttötilannetta, ja tiedot tulevan viikon poissaolijoista on käytävä erikseen selaamassa kyseisen viikon kohdalta. Riskeinä ovat myös Excel-tiedostojen häviäminen tai turmelu. Excel-tiedostoja on myös luotava uudelleen, niihin on lisättävä aina erikseen viikonloput ja arkipyhät, eivätkä ne tarjoa ilmoituksia tulevista loma-ajoista.

2.2 Tavoitteet

Sovelluksen määrittely tehtiin tilaajayrityksen työryhmän esimiehen kanssa. Lomasovelluksen avulla työryhmässä voidaan varmistaa, että työntekijöiden lomat limittyvät niin, että työryhmässä on aina riittävästi henkilökuntaa paikalla. Sovelluksesta tulisi tehdä kevyt, nopeasti yksin kehitettävä, edullinen ja helposti ylläpidettävä. Sovellus toteutetaan englanniksi, eikä siihen toteuteta toistaiseksi kirjautumista. Tilaaja pyysi huomioimaan EU:n GDPR-lain henkilötietojen käsittelyssä [2], minkä vuoksi sovelluksessa ei voida käyttää suoraan työntekijöiden nimiä, vaan esimerkiksi etunimiä.

Kun työntekijä valitsee suunnitellun loman ajankohdan, sovellus voisi näyttää reaaliaikaisesti lomalla olijoiden määrän valitulla ajankohdalla. Sovelluksen avulla voitaisiin myös tarkastella eri ajanjaksojen lomatilannetta yleisellä tasolla. Yleisen lomatilannenäkymän tarkka toteutus jäi vielä määrittelykeskusteluissa tilaajan kanssa avoimeksi.

Tekijä suunnitteli eri vaihtoehtoja näkymään, kuten kalenterimuotoisen lämpökartan, jossa näkyisi päivittäin eri värein, kuinka monta lomalla olijaa kyseisenä päivänä on. Tämä olisi anonyymi ratkaisu, joka ei paljastaisi yksittäisen lomailijan loma-aikoja muille. Toinen vaihtoehto olisi kuukausilista, jossa olisi merkattuna riveittäin kuukauden lomalla olevat työntekijät ja työntekijöiden lomien sijoittuminen kalenteripäiviin. Kuukausilista olisi lämpökarttaa parempi ratkaisu, koska yksittäisten henkilöiden loma-aikoihin voidaan puuttua. Huonona puolena on, että käyttäjät saattavat olla tunnistettavissa ilman kirjautumista olevalla sivustolla.

On helppo nähdä, että tulevat käyttäjät (työryhmän työntekijät) tulevat vertaamaan lopullista lomasovellusta nykyiseen Excel-ratkaisuun. Sovelluksen tavoitteiksi on siis asetettava yksinkertaisuus, toimintavarmuus ja helppo loman lisäys, jotka ovat selkeitä Excel-tiedostojen etuja. Lisäetuna lomasovellukselle on, että työryhmän viestijärjestelmään, kuten Slackiin, voitaisiin luoda automaattiset ilmoitukset seuraavan viikon lomalla olijoiden määrästä.

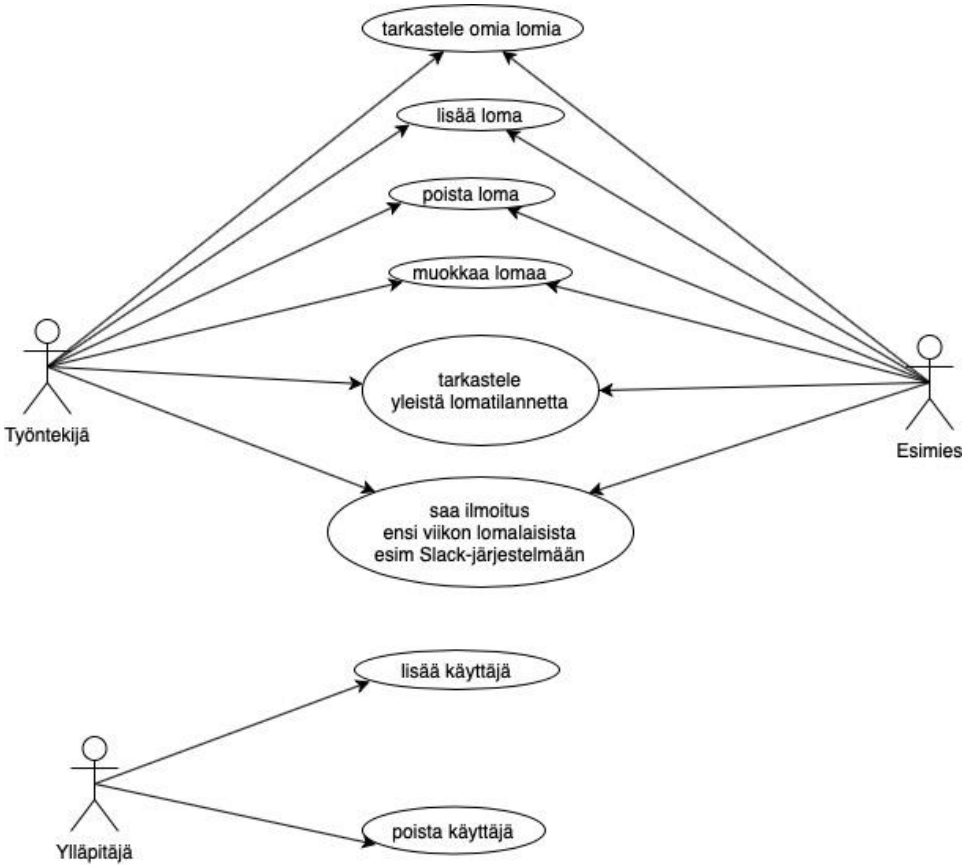
Ilmoituksia ja jatkokehitystä varten sovellukseen toteutetaan ohjelmointirajapinta, jonka avulla järjestelmään voidaan tulevaisuudessa esimerkiksi integroida muitakin käyttötapoja.

Lisäominaisuuksina sovellus voisi ilmoittaa, mikäli työntekijä ei ole vielä täyttänyt loma-aikatauluaan määräaikaan mennessä tai antaa työnjohdolle mahdollisuutta hyväksyä ja hylätä lomia. Sovelluksella voitaisiin myös ylläpitää työntekijöiden lomapäivien määrää ja lomatietokannasta olisi mahdollista pitää varmuuskopiota.

2.3 Käyttötapaukset ja sovelluksen rakenne

Sekä työntekijällä että esimiehellä on samantapaiset sovelluksen käyttötarpeet (kuva 1). Molempien on pystyttävä lisäämään, tarkastelemaan, muokkaamaan ja poistamaan omia loma-aikojaan sekä saamaan ilmoitus seuraavan viikon lomalla olijoiden määrästä. Molemmat voivat tarkastella yleistä lomatilannetta eli montako henkilöä on eri ajanjaksoina lomalla. Tämä yleinen lomatilanne voisi heijastua myös loma-aikojen päivämäärien valintavaiheessa, ja sovellus voisi varoittaa, jos valitulla ajankohdalla on liikaa lomaansa viettäviä työntekijöitä. Lisäksi ylläpitäjän tulisi pystyä poistamaan ja lisäämään käyttäjiä sekä

muokkaamaan sovelluksen asetuksia, kuten työntekijöiden kokonaismäärää (kuva 1).



Kuva 1. Käyttötapauskaavio.

Sovellukseen tulisi siis sisältyä loma-aikojen CRUD-näkymä (Create, Read, Update, Delete), jossa lomia voidaan lisätä, tarkastella, muokata ja poistaa. Lisäksi sovelluksessa pitäisi olla näkymä yleisestä lomatilanteesta sekä ylläpitäjän näkymä.

3 Lomasovelluksen suunnittelu

3.1 Vaihtoehdot teknologiapinoksi

Lomasovellukselle on valittava teknologiapino (stack) eli dataekosysteemi, johon kuuluvat sovelluksen rakentamiseen ja käyttöön liittyvät työkalut,

kehykset ja kirjastot. Pino kuvaa sovelluksen arkkitehtuurin kerroksittaisena rakenteena, johon voi kuulua esimerkiksi käyttöjärjestelmä, web-palvelin, tietokanta ja ohjelmointikieli. Teknologiapino voidaan valita erikseen sovelluksen asiakaspuolelle (frontend) sekä palvelinpuolelle (backend) tai valita fullstack-teknologiapino, joka sisältää molemmat. Teknologioiden suosio vaihtelee, ja vain osa jää vuosikymmeniksi käyttöön, kuten Apachen web-palvelin. Toisaalta alalle syntyy jatkuvasti uusia ja suosittuja teknologioita, jotka saattavat toimia vanhoja ja totuttuja ratkaisuja tehokkaammin. Riippuu tilanteesta, mikä teknologia on oikea valinta. Ohjelmistokehittäjillä voi olla ennakkoluuloja eri teknologioita kohtaan, mutta teknologioita on syytä tarkastella objektiivisesti käyttötarpeiden mukaan.

Arunin mukaan pinon valinta on tärkeää, koska se vaikuttaa suoraan sovelluksen toimintakykyyn nyt ja tulevaisuudessa, skaalautuvuuteen sekä kapasiteettiin. On myös otettava huomioon, kehitetäänkö sovellus pilvipalveluun vai omalle palvelimelle. On siis tärkeää tehdä pinoista tarkka analyysi ennen teknologioiden valintaa. [3.] Ennen sovelluksen pinon valintaa on syytä vertailla muutamien suosittujen teknologiapinojen vahvuuksia ja heikkouksia.

3.1.1 MEAN-, MERN- ja MEVN-pinot

MEAN-pinoon kuuluvat MongoDB-tietokanta, Express.js-kehys (Node.js), Angular-kehys (JavaScript) ja Node.js-suoritusympäristö. Angular korvataan MERN-pinossa React-kirjastolla (JavaScript) ja MEVN-pinossa Vue-kehyksellä (JavaScript). Marijanin mukaan näiden kaikkien pinojen etuna on mahdollisuus koodata koko pino JavaScriptilla, mikä mahdollistaa koodin uudelleenkäytön sekä frontendissä että backendissä. Hyviä puolia ovat myös ilmaisuus, sopivuus pilvipalveluihin (joustavuus ja laajennettavuus) ja tietokannan skaalautuvuus. [4.] MongoDB on NoSQL-tietokanta, jossa datan mallintaminen ei vaadi relaatiotietokantojen tapaan erillisiä taulukoita ja niiden yhdistämistä join-lausekkeilla. Lisäksi skeemat ovat joustavia. [5.]

Marijanin mukaan nämä pinot eivät myöskään ole riippuvaisia yhdestä käyttöjärjestelmästä, mutta niiden huonoina puolina ovat huonompi soveltuvuus laajoihin sovelluksiin, sivujen hitaampi ladattavuus (JavaScript) ja MongoDB:n vähäisempi käyttövarmuus relaatiotietokantoihin verrattuna. [4.] JavaScript on suosittu ohjelmointikieli tällä hetkellä, mutta kehysten nopea kehitys voi aiheuttaa ylläpidolle haasteita, ja monet teknologiat voivat myös hävitä nopeasti [6].

Kapoorin mukaan MERN-pinossa on muutamia selkeitä etuja: se perustuu avoimeen lähdekoodiin, joten sitä kehitetään jatkuvasti, eikä se sido käyttäjää tietyn ohjelmiston toimittajaan. MERN:iin on myös olemassa ilmaisia template-malleja ja valmiita kehyksiä, jotka säästävät aikaa ja tuovat joustavuutta. MERN-pinon teknologiat ovat myös helppoja oppia, koska ne ovat hyvin dokumentoituja, suosittuja ja minimalistisia. [7.] MERN-pinoon on olemassa myös hyviä ilmaisia fullstack-kursseja, kuten Helsingin yliopiston Full Stack Open.

3.1.2 LAMP- ja LEMP-pino

LAMP on yksi ensimmäisiä avoimen lähdekoodin pinoista, joka on edelleen käytössä. LAMP koostuu Linux-käyttöjärjestelmästä, Apache-verkkopalvelimesta, MySQL-tietokannasta ja PHP-ohjelmointikielestä (kuva 2). PHP voidaan myös korvata Perlillä tai Pythonilla. [3.] Aiemmin Apache oli hyvin suosittu verkkopalvelin, esimerkiksi vuonna 2005 kaikista web-sivuista 71 % käytti Apachea. Sen suosio on kuitenkin laskenut tasaisesti, ja maaliskuussa 2022 sen osuus kaikista websivuista on 23 %. [8.] Apache koostuu moduuleista, joiden avulla se toimii yksi- tai monisäikeisesti. Stevensin mukaan tämä arkkitehtuuri tuo joustavuutta ja muokkautuvuutta. [9.]

Kuva 2. LAMP-pino: Linux, Apache, MySQL ja PHP.

LAMP-pinon hyviä puolia ovat hyvä yhteisötuki, yleisyys sekä helppo komponenttien korvattavuus ja asennus. MySQL on myös luotettava ja PHP helppo oppia, mutta toisaalta PHP:n täysi hallinta on vaikeaa. [6.] Huonoja puolia ovat myös Linux-sidonnaisuus, MySQL-relaatiotietokannan huonompi tehokkuus ja Apachen suoritusongelmat suurilla kuormilla [10]. Linux on hyvä vaihtoehto käyttöjärjestelmäksi, koska Unixin kaltaisena käyttöjärjestelmänä se on tehokkain vaihtoehto [9], mutta Linux voidaan myös korvata Windowsilla (WAMP-pino), Macilla (MAMP-pino) tai käyttää käyttöjärjestelmästä riippumatta (XAMPP-pino).

LEMP-pinossa Apache on korvattu NGINX-palvelimella, joka ei käytä pyyntöjen käsittelyyn Apachen tapaan säikeitä, vaan tapahtumapohjaista, skaalautuvaa ja asynkronista (eli ei-reaaliaikaista) arkkitehtuuria. NGINX:llä on korkea suorituskyky sekä nopeus, ja se pystyykin käsittelemään tuhansia HTTP-yhteyksiä samanaikaisesti (HTTP on selainten ja web-palvelinten tiedonsiirtoprotokolla). NGINX ei siten rasita muistia ja prosessoria. NGINX on suosittu valinta myös mikropalveluita varten. Toisaalta NGINX voi olla vaikeampi asentaa ja integroida kuin Apache. NGINX suoriutuu hyvin staattisen web-sisällön siirrossa, mutta dynaamisen sisällön siirtoon tarvitaan lisäprosessori. [9.] Netcraftin uudesta verkkopalvelinselvityksestä selviää, että NGINX on kasvattanut suosiotaan tasaisesti ja on muutamilla mittareilla tämän hetken suosituin verkkopalvelin 31 % osuudellaan kaikista sivustoista [8]. Apachen etuna on hyvä integroitavuus PHP:hen ja sitä kautta dynaamiseen sisällön siirtoon [9].

3.1.3 WISA-pino

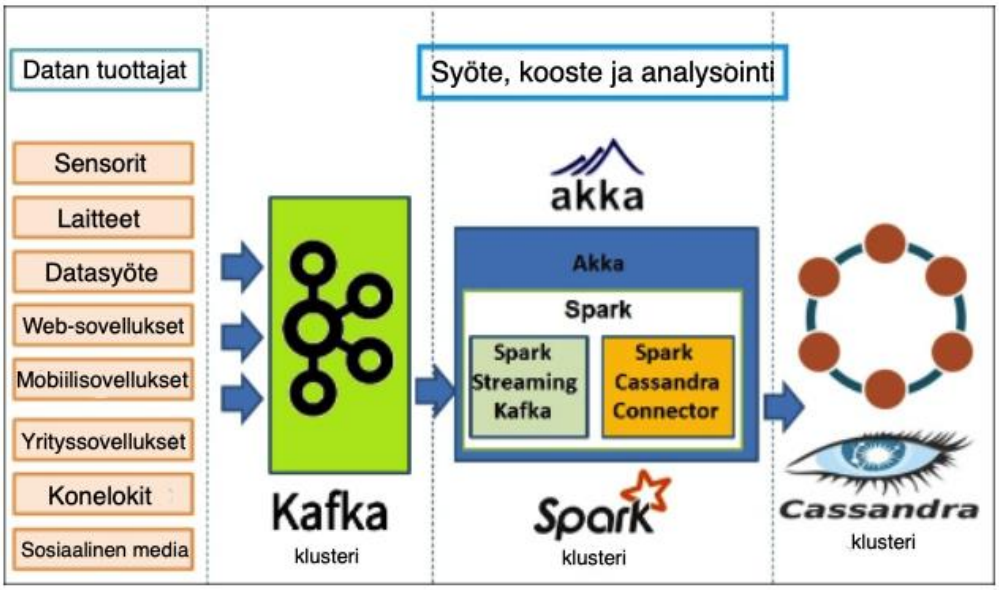
WISA on Microsoftin web-ratkaisu, jossa käytetään Windows-palvelinta, Internet Information Services (IIS) -verkkopalvelinta, SQL Server -relaatiotietokantapalvelinta ja ASP.NET-verkkokehystä [11]. ASP.NET on osa .NET-kehystä [12].

.NET-kehys on ilmainen Microsoftin ohjelmistokehys ja -ekosysteemi, jolla voi luoda laajalti verkko- ja pilvipalveluita sekä työpöytäsovelluksia. Web-palveluiden luomiseen käytetään ASP.NET-kehystä, jonka etuna on automaattinen monitorointi. Se varmistaa paremman vakauden ja läpinäkyvyyden ilmoittamalla esimerkiksi muistivuodoista. Uuden version (.NET Core) ASP.NET Core on modulaarinen kehys, joka on alustariippumaton. [12.]

Ohjelmointikieliä .NET:issä ovat C#, F# ja Visual Basic. Kehyksen hyviä puolia ovat olio-ohjelmointikoodin uudelleenkäytettävyys, Visual Studio -ohjelmointiympäristö, helppo valmiin sovelluksen julkaisu (deployment), laaja yhteisötuki sekä luotettava ja yksinkertainen välimuistijärjestelmä. Huonoja puolia ovat olio-ohjelmointimallista johtuva keskittyminen olioihin toimintojen sijaan, lisenssimaksut, muistivuodot ja kunnollisen dokumentaation puuttuminen. Myös kehyksen Microsoft-sidonnaisuus vaatii parhaaseen integraatioon Windows-käyttöjärjestelmän ja pilvijärjestelmäksi Microsoftin Azuren. [12.]

3.1.4 SMACK-pino

SMACK-pinoon (kuva 3) kuuluu Apache-järjestön tuotteet Spark-engine, Mesos-ydin, Cassandra-tietokanta sekä Kafka-viestinvälittäjä. Lisäksi pinoon kuuluu Akka-kirjasto/ajoympäristö. Apache Spark on laajaskaalainen datan prosessointiympäristö. [13.] Mesos toimii ytimenä hajautetuille järjestelmille tai klusterialustana [14]. Mesosilla on monia hyödyllisiä kehyksiä, kuten Marathon, jolla voidaan automaattisesti skaalata klusteria, eli se toimii Kubernetesin tapaan. Cassandra on suorituskykyinen tietokanta, joka on helppo liittää Sparkiin ja Mesosiin. Kafka on tehokas hajautettu viestintäjärjestelmä, joka toimii esimerkiksi sovellustasolla datan siirtäjänä seuraaville pinon tasoille. Akka on viestipohjainen ja asynkroninen kirjasto ja ajoympäristö Java-virtuaalikoneen päälle rakennettavien sovellusten rakentamiseen Javalla ja Scalalla. [13.]



Kuva 3. SMACK-arkkitehtuuri, suomennettu lähteestä [13].

SMACK-pinon osat ovat avointa lähdekoodia ja ne on helppo integroida. SMACK-pino on skaalautuva ja matalaviiveinen, ja se on hyödyllinen esimerkiksi big data -ohjelmistoissa, joissa käsitellään erittäin suuria datamääriä. [14.] Huonoina puolina SMACK-pino vaikuttaa jääneen melko vähällä huomiolle ja Mesos Maraton on vähemmän tunnettu kuin Kubernetes. Lomasovellus ei myöskään käsittele big dataa, joten SMACK-pinon edut eivät hyödynnä projektia.

3.1.5 JAMStack-pino

JAMStackilla voidaan tehdä staattisia verkkosivuja hyödyntäen JavaScriptia, ohjelmointirajapintoja (API) ja merkintäkieltä (markup). Merkintäkieli on HTML- ja CSS-koodia, jonka päälle dynaamisuus syntyy JavaScriptilla, joka hakee backendina toimivista ohjelmointirajapinnoista dataa. [15.] JavaScript voi olla esimerkiksi puhdasta JavaScriptiä tai JavaScript-kehys, ja merkintäkielessä voidaan käyttää koontiohjelmistoa, kuten Grunt tai staattisten web-sivujen luontiohjelmistoa, kuten Jekyll [16]. Täten backend on mikropalveluarkkitehtuurin mukainen. Mahdollisimman suuri osa HTML:stä on

valmiina ja tallennettuna sisällönjakeluverkkoon (content delivery network, CDN), mikä tekee sivuista nopeasti latautuvia [15].

JAMStack on skaalautuva käytön suhteen, ja frontend-painottumisen vuoksi ohjelmoija voi keskittyä käyttökokemukseen backendin sijaan [15]. JAMStack on myös turvallinen, koska pinosta puuttuu tietokanta. Jatkuva julkaisu (continuous deployment) onnistuu helposti, koska koko koodi voidaan pitää git-versionhallinnassa, jonne tehdyt muutokset voivat käynnistää uuden julkaisun. Avointen ratkaisujen määrä kasvaa ajan myötä ja käytettävän API:n voi myös rakentaa itse. JAMStack ei sovellu hyvin interaktiivisiin ominaisuuksiin tai dynaamisuutta korostaviin sivustoihin. Dynaamisuutta voidaan lisätä valmiilla komponenteilla, mutta tämä on JAMStackin hyötyjen vastaista. [16.]

3.1.6 JavaScript, Java ja Spring-kehys

JavaScript-kehyksiä ja -kirjastoja (kuten React tai Angular) voidaan käyttää myös Javan kanssa. Spring on Javan avoin sovelluskehityskehys, jolla voi tehdä paitsi Java-ohjelmia, myös web-sovelluksia Java EE -alustalle laajennusten avulla [17]. Heiken mukaan tämä on enemmänkin isompien yritysten kuin itsenäisten kehittäjien suosima teknologiavalinta monimutkaisuutensa vuoksi [6].

Spring on kevyt kehys, joka mahdollistaa modulaarisen arkkitehtuurin ja löyhät kytkökset. Spring Boot on avoimen lähdekoodin kehys, joka helpottaa Spring-sovelluksen tekemistä. Spring Bootilla Spring-sovellus voidaan konfiguroida automaattisesti ilman jatkuvasti uudelleen toistettavaa (boilerplate) koodia. Sen kanssa voidaan käyttää eri tietokantoja ja viestijonopalveluita ja sen mukana tulee valmiina HTTP-palvelin, kuten Tomcat. Pinon hyviä puolia ovat Javan suosio, Spring Bootin suuri yhteisö ja hyvä ilmainen opiskelumateriaali. Pinon huonoja puolia ovat tarpeettomasti asentuvat lisäosat sekä vanhentuneen koodin päivityksen hankaluus. [18.]

3.1.7 Muut pinot

Muita vaihtoehtoja ovat esimerkiksi Ruby on Rails -verkkokehys, joka perustuu Ruby-kieleen. Rails on nopeaa oppia ja käyttää, ja sillä on aktiivinen kehittäjäyhteisö ja satoja kirjastoja. Huonoina puolina ovat heikompi suorituskyky esimerkiksi Node.js:ään verrattuna, kehyksen joustamattomuus elementtien suhteen sekä kielen jatkuvat muutokset. Koodausvirheet voivat myös vaikuttaa suuresti suorituskykyyn. [19.]

Pythonilla ja siihen perustuvalla Django-kehyksellä voidaan rakentaa web-sovelluksia Ruby on Railsin (RoR) tapaan DRY-periaatteella (Don’t repeat yourself) eli pyritään välttämään toistoa koodirakenteissa (boilerplate). Djangon hyviä puolia on RoR:n tapaan nopeus ja yhteisötuki. Etuja ovat myös kattava dokumentaatio, turvallisuus sekä joustavuus erilaisiin sovelluksiin. Huonona puolena on Djangon sopimattomuus yksinkertaisiin sovelluksiin, jolloin Python-pohjainen Flask voisi toimia paremmin. RoR:n tapaan Djangolla voi olla heikko suorituskyky, ja kehyksen laajuus vaatii sovelluksen koko rakenteen suunnittelun etukäteen. [20.]

Myös serverless-ratkaisuja on olemassa, joissa pilvipalvelun tarjoaja hoitaa backend-palvelut ja laskuttaa niistä käytön mukaan [21]. Esimerkiksi Azuressa on serverless-ratkaisu Azure Functions [22].

3.1.8 Sovellukseen valittava pino

Määrittelyn mukaan lomasovelluksen halutaan olevan pieni, nopeasti yksin kehitettävä, edullinen ja helposti ylläpidettävä. Pinovaihtoehdot on koottu taulukkoon 1, jossa vertaillaan pinojen hyviä ja huonoja puolia lomasovellusta ajatellen.

Taulukko 1. Teknologiapinovertailu.

Pino	**Teknologiat**	**Edut**	**Haitat**
MEAN, MERN, MEVN	MongoDB, Express.js, Angular/React/Vue, Node.js	Yksi kieli, skaalautuvuus, hyvin dokumentoitu, minimalistinen	Ei sovi laajoihin sovelluksiin, tietokannan käyttövarmuus, kielten muuttuminen
LAMP	Linux, Apache, MySQL, PHP/Perl/Python	Yhteisö, komponenttien korvattavuus, tietokannan luotettavuus	Linux-sidonnaisuus, Apachen suoritusongelmat, PHP:n hallinta
LEMP	Linux, NGINX, MySQL, PHP/Perl/Python	Korkea suorituskyky ja nopeus, sopii mikropalveluihin, tietokannan luotettavuus	Linux-sidonnaisuus, PHP:n hallinta, NGINX:n asennus
WISA	Windows, IIS, SQL Server, ASP.NET	Helppo julkaisu, automaattinen monitorointi, yhteisö, Visual Studio	Lisenssimaksut, dokumentaatio, Microsoft-sidonnaisuus
SMACK	Spark, Mesos, Akka, Cassandra, Kafka	Osien helppo integroituvuus, skaalautuva, big dataan sopiva	Ajantasaisuus, pinon dokumentaatio, vähäinen suosio
JAMStack	JavaScript, HTML, CSS	Nopea sivujen latautuvuus, skaalautuva, turvallisuus, jatkuva julkaisu	Ei tietokantaa, ei paras dynaamisiin web-sivuihin
Spring ja JavaScript-kehys	Java, JavaScript	Modulaarisuus, automaattinen konfigurointi, valmis palvelin, yhteisö, ilmainen opiskelumateriaali	Monimutkaisuus, vanhentuneen koodin päivityksen vaikeus, tarpeettomat lisäosat

Pino	Teknologiat	Edut	Haitat
Ruby on Rails	Ruby	Nopea, aloittelijaystävällinen, yhteisö, kirjastot	Suorituskyky, joustamattomuus, kielen muuttuminen
Django	Python	Nopea, yhteisö, dokumentaatio, joustavuus sovelluksen suhteen	Suorituskyky, ei sovi yksinkertaisiin sovelluksiin, vaatii tarkan suunnittelun

Spring-, WISA-, SMACK- ja Django-pohjaiset ratkaisut vaikuttavat liian raskailta yksinkertaiseksi määriteltyyn sovellukseen. JAMStack ja serverless vaikuttavat uudenlaisilta ja mielenkiintoisilta ratkaisuilta, mutta sovellukseen on suunniteltu dynaamisuutta, mahdollisuutta hallita palvelinpuolta ja tietokantaa. Ruby on Rails vaatisi uuden ohjelmointikielen oppimista, mikä aiheuttaa haastetta projektin aikataulun vuoksi. Parhaiksi vaihtoehdoiksi nähdään MERN ja LAMP suosionsa, keveytensä sekä tekijän ja työryhmän osaamisen perusteella, joten verrataan näitä kahta pinovaihtoehtoa.

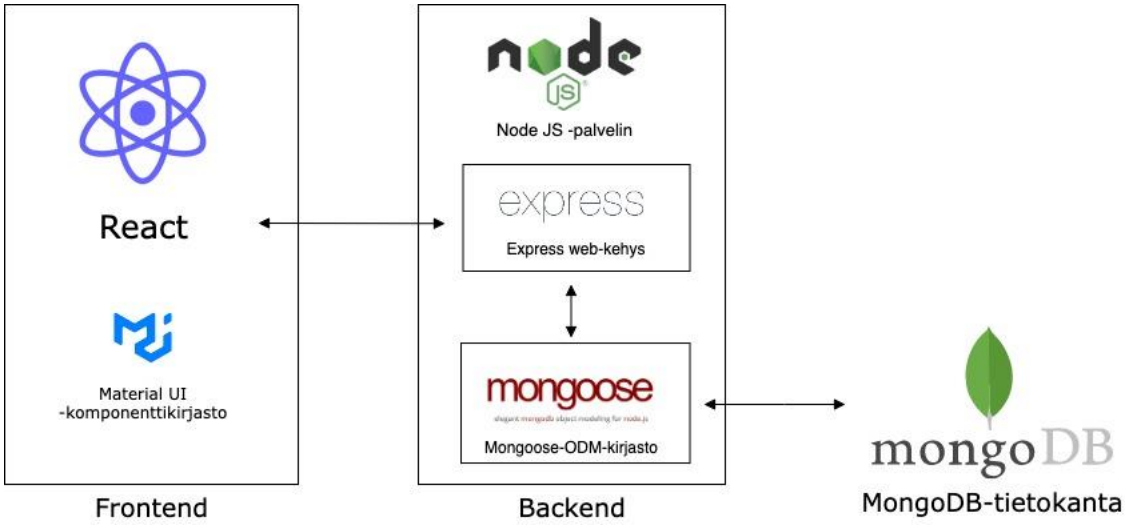
MERN-pinolla sovellus voitaisiin koodata kokonaisuudessaan JavaScriptillä, joten ohjelmoijan ei tarvitse opetella muita ohjelmointikieliä, kun taas LAMP vaatisi osaamista PHP- ja SQL-kielistä. React ja JavaScript ovat suosittuja tällä hetkellä, sillä Stack Overflow’n kehittäjäkyselyssä 2021 React oli suosituin web-kehys ja JavaScript suosituin ohjelmointikieli 9. vuotena peräkkäin [23]. MERN soveltuu hyvin dynaamisiin verkkokäyttöliittymiin ja kalentereihin, mikä on etu lomasovellusta ajatellen [24]. MERN ei ole parhaimmillaan laajoissa sovelluksissa, mutta lomasovellus on pieni [4].

MERN:in etu on myös NoSQL:n tehokkuus ja joustavuus verrattuna relaatiotietokantaan [10], sillä sovelluksen käsittelemän datan rakenne voi muuttua alustavan toteutuksen aikana ja sen jälkeen. Pinon valinnassa on otettava huomioon myös tekijän ja työryhmän osaaminen sekä ohjelmointikielien suosio tällä hetkellä. Ohjelmointikielen valinta vaikuttaa

sovelluksen ylläpitoon tulevaisuudessa. JavaScript on työryhmässä yleisesti tunnettu, ja MERN vaatisi siten vähemmän uuden opettelua myös muilta työryhmän jäseniltä. Lomasovelluksen pinoksi on näiden syiden vuoksi valittu MERN.

3.2 MERN-pino

MERN-pinoon kuuluvat MongoDB-tietokanta, Express.js-kehys, React-kirjasto ja Node.js-suoritusympäristö. Lisäksi projektissa on päätetty käyttää Material UI -käyttöliittymätyylikirjastoa ja mongoose-ODM-kirjastoa (kuva 4). Tarkastellaan MERN-pinoa teknologia kerrallaan.



Kuva 4. Lomasovelluksen MERN-pino. Sovellettu ja suomennettu lähteestä [25].

3.2.1 Node.js ja Express.js

Node.js on käyttöjärjestelmäriippumaton JavaScriptin suoritusympäristö, jonka etuina ovat esimerkiksi npm-paketinhallintajärjestelmä ja yksisäikeisyyden myötä tehokkuus. Npm:stä on mahdollista ladata satojatuhansia uudelleenkäytettäviä pakettitiedostoja ja sillä voidaan myös automatisoida kehitystyövaiheita. Node.js:n hyviä puolia on aktiivinen kehittäjäyhteisö ja mahdollisuus koodata JavaScriptilla myös backend. [26.] Se on myös hyvin

dokumentoitu ja sen asynkroninen tapahtumasilmukka mahdollistaa skaalautuvuuden ja tehokkuuden. JavaScript mahdollistaa rinnakkaisuuden, eikä Node.js:ssä ole käännösvaihetta. Huonoja puolia ovat turvallisuusaukot yksisäikeisyyden myötä sekä suuri tilan- ja usein myös muistintarve. [7.]

Express.js on Node.js:n web-kehys, jolla voi koodata helposti esimerkiksi HTTP-pyyntöjen käsittelijöitä [26]. Express.js mahdollistaa myös suuren määrän väliohjelmistoja (middleware) muun muassa evästeiden tai istuntojen hallintaan [27]. Express.js:n etuja ovat saman ohjelmointikielen käyttö läpi sovelluksen sekä JavaScriptin kehittymismahdollisuudet. Toisaalta Express.js-kehystä voi käyttää ainoastaan Node.js:n kanssa, eivätkä tietyt kehykset sovi sen kanssa yhteen, mikä voi johtaa muistivuotoihin tai kaatumisiin. [7.]

3.2.2 React

React on alun perin Facebookin kehittämä deklaratiivinen JavaScript-kirjasto käyttöjärjestelmien luomiseen [28]. Reactin avulla käyttöjärjestelmä kootaan komponenttiluokista tai -tyypeistä. Näille komponenteille voidaan antaa parametrejä (props), ja ne voivat tallentaa tiloja (state). Tilalle asetetaan alkutila, ja kun tila vaihtuu, komponentti renderöityy uudelleen. [29.]

React-komponentit palauttavat render-metodilla kuvauksen lopullisesta näkymästä React-elementtinä. Tähän käytetään yleisesti JSX-syntaksia, joka helpottaa React-elementtien kirjoitusta ja jolla voidaan kirjoittaa suoraan myös JavaScript-koodia. Esimerkiksi kuvassa 5 button-syntaksi muutetaan build-vaiheessa muotoon React.createElement (’button’). Yksinkertaisempi tapa kirjoittaa komponentteja ovat funktiokomponentit, joilla ei ole omaa tilaa ja joka sisältää vain render-metodin return-funktion (kuva 5). [29.]

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

Kuva 5 Funktiokomponentti Reactissa [29].

Reactin versio 16.8 toi mahdollisuuden käyttää koukkuja (hooks). Koukkujen avulla voidaan uudelleenkäyttää tilallista logiikkaa ilman luokkia. [30] Esimerkiksi tilanhallintaan uudelleen renderöitymisten välillä käytetty useState-koukku palauttaa parin: tilan nykyisen arvon ja funktion, jolla tila päivitetään. UseStatelle asetetaan argumenttina tilan alkuarvo, jota käytetään vain ensimmäisessä renderöinnissä. Toinen hyödyllinen koukku on useEffect, jonka avulla voidaan aiheuttaa renderöinnin ulkopuolisia sivuvaikutuksia (side effects) muihin komponentteihin. Kun muutokset on tehty sivun DOM:iin, useEffect-koukkua kutsutaan oletusarvoisesti jokaisella renderöinnillä. [31.]

Kapoorin mukaan Reactin hyviä puolia ovat nopea renderöinti, joustavuus ja selkeys. Huonoja puolia ovat virheiden etsimisen ja poiston vaikeus (itsenäiset komponentit) sekä alkuperäisen kehittäjän (Meta/Facebook) tuen puute. [7.] Lopullinen sovellus koostuu käyttöliittymäkomponenteista, kuten napeista, ponnahdusikkunoista ja layoutin alueista. Reactia varten on olemassa monia hyödyllisiä käyttöliittymätyylikirjastoja, kuten React Bootstrap ja Material-UI. Lomasovelluksessa suunnitellaan käytettäväksi valmista käyttöliittymätyylikirjastoa, koska ne ovat nopeita, helppokäyttöisiä ja yhteensopivia [32]. Projektiin valittu kirjasto on Material-UI, koska se on monipuolinen, tuttu entuudestaan tekijälle ja vaikuttaa aktiivisesti ylläpidetyltä.

3.2.3 MongoDB ja mongoose

MongoDB-tietokantojen kyselyt ovat nopeampia kuin relaatiotietokannoissa, ja MongoDB:n tietomallit ovat helposti muokattavia [33]. MongoDB:ssä

tallennusmuotona on dokumentti (document), joka on JSON-objektin kaltainen avain-arvorakenne. Dokumentit tallennetaan kokoelmiin (collections), jotka vastaavat relaatiotietokantojen tauluja. [34.] Datan siirtoon voidaan käyttää JSON-tiedostomuotoa, jonka tallentamiseen MongoDB on kehitetty [24]. MongoDB myös tukee eri tietotyyppejä, mikä tekee monimutkaistenkin datarakenteiden mallintamisen mahdolliseksi. MongoDB:n huonoja puolia on uudenlaisen datamallintamisen opettelu sekä dokumentin pieni maksimikoko (16 MB). [7; 35.]

MongoDB-tietokannan dataa voidaan aggregoida eli koostaa omalla syntaksillaan, mikä vastaa esimerkiksi SQL-kyselyitä MySQL:ssä. Aggregointi tapahtuu putkittamalla työvaiheita (stage). Esimerkiksi $project-vaihe sisällyttää tai jättää sisällyttämästä tietyt tietokentät, ja $match-vaihe suodattaa vain dokumentit, jotka läpäisevät tietyn ehdon. Muita hyödyllisiä vaiheita ovat esimerkiksi $unwind, joka purkaa dokumentin unwind-parametrin mukaan useammaksi dokumentiksi, joissa unwind-parametri saa kaikki mahdolliset arvonsa. $group-vaihe ryhmittää dataa _id-parametrinsa mukaisesti niin, että jokainen ryhmä saa oman _id-parametrin arvonsa. Jos _id on null, tila palauttaa lukumäärän. $sort-vaihe järjestää dokumentit tietyn ehdon mukaisesti. [35.]

Mongoose on Node.js-pohjainen Object Data Modeling (ODM) -kirjasto MongoDB:lle. ODM on Object Relational Mapperin (ORM) kaltainen tekniikka. ORM:in avulla dataa muunnetaan tyyppijärjestelmien välillä käyttämällä olio-ohjelmointikieliä. Esimerkki ORM-ratkaisusta on Javan Hibernate, jolla voidaan kuvata (map) oliomalli relaatiotietokantaan. Mongoosen avulla voidaan mallintaa sovelluksen dataa skeemojen avulla sekä muutenkin helpottaa MongoDB-tietokannan käyttöä. Mongoosen huonoja puolia on, että skeema on olemassa vain Node.js-ympäristössä, eikä itse MongoDB siis tunne skeemaa, jos tietokantaa käytetään muuta kautta. [36.]

3.3 Ohjelmointirajapinta ja API-protokolla

Lomasovellukseen suunnitellaan API eli ohjelmointirajapinta, jonka kautta voidaan siirtää tietoa esimerkiksi web-sivun ja tietokannan välillä. API on sopimus, jonka mukaisesti kaksi järjestelmää voivat kommunikoida. API siis helpottaa integraatioiden, kuten automaattisten Slack-viestien, lisäämisen sovellukseen myöhemmin. API on tekninen määrittely, joka kuvaa tiedonsiirtovaihtoehtoja ratkaisujen välillä ja jonka muoto perustuu tiedonprosessointi- ja tiedonsiirtopyyntötapoihin sekä tämän määrittelyn mukainen ohjelmisto. Dokumentointi onkin olennainen osa API:n luomista ja sen kirjoittamiseen on olemassa myös työkaluja, kuten Swagger tai Postman. API:n dokumentaatio sisältää funktiot, luokat, palautuvan datatyypin ja argumentit. [37.]

API sisältää funktiokutsuja, jotka koostuvat verbeistä ja substantiiveista, kuten ”lisää loma” tai ”anna käyttäjä”. Lomasovellusta varten on valittava API-protokolla, joka kuvaa tiedonsiirtotavan API:n ja sitä käyttävän palvelun välillä. [37.] API on suunniteltava paitsi sovelluksen, myös jatkosuunnittelun tarpeiden mukaisesti. Vaihtoehtoja sovelluksen API-protokollaksi ovat esimerkiksi SOAP, gRPC, REST ja GraphQL.

3.3.1 SOAP ja gRPC

SOAP (Service Object Access Protocol) on Microsoftin kehittämä kevyt protokolla jäsentyneen datan siirtoon hajautetussa ympäristössä. Se sisältää pyyntö- ja vastausviestien syntaksin ja se käyttää HTTP:ta ja SMTP:ta (sähköpostipalvelimien protokolla viestinvälitykseen). SOAP:issa käytetään XML:ää, joka on sekä koneen että ihmisen luettavaa merkintäkieltä. SOAP:ia käytetään yleisesti yrityssovelluksissa, esimerkiksi web-pohjaisissa maksuprosesseissa sekä identiteetti- ja asiakkuudenhallinnassa turvallisuutensa vuoksi. [37.]

gRPC on Googlen kehittämä, uudehko API-kehys, joka käyttää JSON:in tai XML:n sijasta protokollapuskureita datan sarjallistamiseen binäärimuotoon. Siinä voidaan käyttää vapaita funktiokutsuja, eikä se ole rajoittunut HTTP-metodeihin. Sarjallistaminen tapahtuu datarakenteen määrittelyllä ja puskurikääntäjän datayhteysluokkien luomisella. gRPC mahdollistaa eri ohjelmointikielien käytön ja hyvän suorituskyvyn, joten se on käytössä etenkin mikropalveluissa. [37.] gRPC:n etuja ovat kevyet viestit ja datan striimaustuki, mutta haittoja ovat yhteisötuen vähäisyys, rajoitettu selaintuki, oppimisen vaikeus ja datan ei-luettava muoto [38].

3.3.2 REST ja GraphQL

REST tarkoittaa arkkitehtuurityyliä ”representational state transfer”. REST on joustava kehys, toisin kuin esimerkiksi SOAP, sillä se on määritelty vain seuraavien periaatteiden pohjalta. Kaikkiin samoihin resursseihin liittyvien API-pyyntöjen tulisi näyttää samanlaisilta ja resurssi, esimerkiksi käyttäjän nimi, kuuluu tiettyyn URI-osoitteeseen. Palvelin ja asiakas ovat itsenäisiä: asiakas tuntee vain resurssin URI-osoitteen, eikä se voi olla vuorovaikutuksessa palvelimeen muilla tavoin. REST API:t ovat tilattomia, eli itse pyynnön on sisällettävä kaikki sen prosessointiin liittyvä tieto, eikä palvelin saa tallentaa mitään pyyntöön liittyvää tietoa. REST on oltava välimuistillinen asiakkaan tai palvelimen puolelta. REST API suunnitellaan kerrokselliseksi, jolloin siinä voi olla välittäjiä. Asiakas ja palvelin eivät tiedä, kommunikoivatko ne suoraan lopullisen ohjelman vai välittäjän kanssa. [39.]

REST API kommunikoi HTTP-pyyntöjen avulla suorittaakseen datan CRUD-toimintoja (luonti, luku, päivitys ja poisto). Kaikkia HTTP-metodeja voidaan käyttää API-kutsuissa (esimerkiksi GET lukuun, POST luomiseen, PUT/PATCH päivittämiseen ja DELETE poistamiseen). Resurssin esitysmuoto (resource representation) kuvaa resurssin tilan, ja se voi olla lähes missä tahansa tiedostomuodossa, kuten JSON, HTML, PHP tai teksti. JSON on suosittu vaihtoehto, koska se on koneen ja ihmisen luettavissa ja riippumaton ohjelmointikielestä. [39.]

REST API:ssa pyynnöt lähetetään tyypillisesti useampaan pääteosoitteeseen. Esimerkiksi lomia voitaisiin hakea osoitteesta /vacations ja käyttäjiä osoitteesta /users. REST-pyyntöjen ongelmia on esimerkiksi se, että tietoa haetaan liikaa tai liian vähän. Jos tarvitaan esimerkiksi vain käyttäjien id-tiedot, voidaan joutua hakemaan kaikki käyttäjiin liittyvät tiedot /users-pääteosoitteesta. Toisaalta tietoa joutuu joissain tilanteissa hakemaan useamman pyynnön avulla. [40.]

REST:in hyvä puoli on, että pääteosoitteet voidaan suunnitella käyttöliittymän näkymien perusteella. Huono puoli on, että frontendia kehittäessä myös backendin vaatimukset muuttuvat tiedon tarpeen muutosten myötä. [40.]

GraphQL on Facebookin luoma kyselykieli, joka luotiin mobiililiikenteen kasvun ja tehokkaamman datansiirtotarpeen vuoksi. GraphQL:n avulla pyynnön tekijä voi määrittää palautuvan datan muodon. Se myös mahdollistaa datan koostamisen eri lähteistä yhdellä API-pyynnöllä. GraphQL:ssä on myös tyyppijärjestelmä. [37.] GraphQL:ssä lähetetään eri datatyyppejä varten tyypillisesti vain yksi pyyntö palvelimelle, joka vastaa pyynnön mukaisesti JSON-objektin [40]. Pyynnöt ovat valmiiksi palautettavan datan muodossa, mikä tekee GraphQL:stä helpon oppia ja käyttää.

Lomasovelluksessa on päätetty käyttää REST-rajapintaa, koska suunniteltu sovellus on yksinkertainen ja sen backendissä pääteosoitteiden määrä ja pyyntöjen monimutkaisuus pysyvät pieninä. Täten GraphQL:stä ei välttämättä ole lisähyötyä. gRPC on parhaimmillaan reaaliaikaisessa viestinnässä ja monikielisessä ympäristössä [38]. Nämä eivät kuvaa suunniteltua lomasovellusta, ja gRPC on myös vaikeampi oppia. SOAP on raskas ratkaisu pieneen sovellukseen ja REST:iin verrattuna hitaampi ja vaikeampi oppia.

3.4 Automaattiset viestit Slack-kanavaan

Slack on monien ICT-yritysten käytössä oleva työviestintäsovellus, jossa keskustelu tapahtuu suorilla viesteillä (valittujen henkilöiden välillä) sekä tiettyä tarkoitusta varten luoduissa kanavissa (ryhmäkeskustelu). Lomasovelluksen

yhtenä tavoitteena on luoda automaattiset ilmoitukset työryhmän Slack-kanavaan seuraavan viikon lomalla olijoista.

Slack-kanaviin voidaan lähettää automaattisia push-viestejä webhookien tai Web API:n chat.postMessage-metodin avulla. Webhookit ovat yksinkertaisempi, mutta vähemmän joustava vaihtoehto ilmoitusten lähettämiseen. Slackiin tehdään App, joka voidaan liittää tiettyyn kanavaan webhookin avulla. [41.] Kun webhookin luo, saa POST-pyynnön lähetystä varten yksikäsitteisen URL-osoitteen. POST-pyynnössä osoitteeseen lähetetään JSON-tietosisältö, joka sisältää viestin esimerkiksi tekstimuodossa (esimerkkikoodi 1). [42.]

```
POST https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXX
Content-type: application/json
{
     “text”: “Hello, world.”
}
```

Esimerkkikoodi 1. POST-pyyntö Slackin webhookiin [42].

Webhookeissa ei tarvita tilanhallintaa tai pääsytietojen autentikointia, mutta toisaalta ne voivat lähettää tietoa vain yhteen kanavaan, eivätkä ne voi lukea tietoa Slackin kanavasta eli ne ovat yksisuuntaisia. URL-osoite on myös pidettävä salassa, koska kuka tahansa voi lähettää sen avulla viestejä määriteltyyn kanavaan. [41.]

Toinen tapa viestien lähettämiseen on Slack Web API:n chat.postMessage-metodi, jonka avulla voidaan lähettää viestejä kaikkiin avonaisiin kanaviin, luoda käyttöliittymiä ja myös vastaanottaa viestejä Slackista. Sen asettaminen on kuitenkin vaikeampaa, koska viestit kulkevat Slack API:n kautta käyttämällä autentikaatio-tokeneita. [41.] Lomasovellusta varten viestin lähetys yhteen suuntaan riittää, joten push-viestin lähettämiseen voidaan käyttää webhookia.

Työntekijöiden on tarkoitus saada Slack-ilmoitus joka viikko, joten tarvitaan tapa automatisoida viestin lähettäminen. Tätä varten on luotava Node.js-palvelimelle cron eli ajastettu komento. Cron(d) on daemon, joka on ajastettu toistumaan tietyllä taajuudella. Daemon on epäinteraktiivinen taustaprosessi, joka odottaa

palvelupyyntöjä. Ajastettuja tehtäviä kutsutaan cron jobeiksi. Unixin kaltaisissa käyttöjärjestelmissä määritellään komennolla ”crontab -e” crontab-tekstitiedosto, jossa määritellään toistuva komento sekä toiston taajuus cronin omalla syntaksilla (esimerkkikoodi 2). [43.]

```
* * * * * komento
┬ ┬ ┬ ┬ ┬
│ │ │ │ │
│ │ │ │ └──── viikonpäivä (0–7) (sunnuntai = 0 tai 7)
│ │ │ └─────── kuukausi (1–12)
│ │ └────────── päivä (1–31)
│ └─────────── tunnit (0–23)
└──────────── minuutit (0–59)
```

Esimerkkikoodi 2. Cron-syntaksi, jossa viisi ensimmäistä merkkiä asettavat komennon toistotaajuuden [43].

Kun sovellus on viety pilvipalveluun, Node.js-palvelimelle voidaan asentaa esimerkiksi node-cron-moduuli, jolle luodaan cron-skripti JavaScriptillä käyttäen cron-syntaksia (esimerkkikoodi 3) [44].

```
const cron = require(“node-cron”)
cron.schedule(“* * * * *”, function () {
     console.log(“running a task every minute”)
});
```

Esimerkkikoodi 3. Node-cron-moduulille luotu komento, joka toistuu minuutin välein [44].

3.5 Aikamäärittelyt JavaScriptissä

Aikamäärittelyt ovat tärkeitä projektin kannalta, koska sovelluksessa tarkastellaan, miten lomien ajankohdat limittyvät kalenteripäivien suhteen. Sovellus pyritään myös suunnittelemaan niin, että sitä voitaisiin käyttää ajasta ja paikasta riippumatta. Ajan käsittely ohjelmoinnissa on hidasta ja aiheuttaa helposti virheitä [45]. Ajan merkitseminen ja tulkinta voivat aiheuttaa epäselvyyksiä. Esimerkiksi päiväyksissä kuukausien ja päivien järjestys vaihtelevat maittain, ja vuosisadan poisjättäminen voi aiheuttaa sekaannusta. Myös kellonajoissa voi olla epäselvyyttä 12 ja 24 tunnin kellojen erojen vuoksi. [46.]

Selainten erojen ja kesäajan vuoksi kriittisissä sovelluksissa olisi hyvä käyttää DateTime-kirjastoja [45], kuten Date-fns ja Moment [47], mutta niidenkin käyttö vaatii ymmärrystä ajan tallentamiseen liittyvistä ongelmista. On myös huomattava kirjastojen rajoitukset, esimerkiksi Moment-kirjastoa ei enää päivitetä [48].

Myös aikavyöhyke on otettava huomioon. ISO 8601 on kansainvälinen standardi ajan esittämiseen. Osien erottajina käytetään päiväyksissä yhdysmerkkiä (-) ja kellonajoissa kaksoispistettä. Standardin mukaan aika voidaan merkata myös ilman erottajia, mutta erottajat helpottavat lukemista. Aika voidaan merkitä vain päiväyksenä, vain kellonaikana tai näiden yhdistelmänä, jolloin päiväys erotetaan kellonajasta T-kirjaimella. On hyvä merkitä aika UTC-aikana, joka merkataan kellonajan loppuun lisättävällä Z-kirjaimella. UTC-aika on koordinoitu yleisaika. Esimerkiksi 2022-02-01T12:00:00.000Z tarkoittaa ajanhetkeä 1. helmikuuta 2022 kello 12 tuntia, 0 minuuttia, 0 sekuntia ja 0 millisekuntia UTC±0 -aikavyöhykkeellä. [46.]

JavaScriptissä on käytössä Date-objekti ajan merkitsemistä varten. Pelkkä Date()-metodi palauttaa merkkijonon ja konstruktori new Date() Date-objektin. [49.] Mozilla ja Moment-kirjaston tekijät eivät suosittele ajan jäsentämistä (parsing) suoraan käyttämällä Date.parse-metodia tai new Date(<string>)-konstruktoria [48; 49].

ISO-standardin mukaisesti merkattu UTC-aika saadaan Date-objektin toISOString-metodilla [45]. ISO 8601:n mukainen aika on kätevä, koska aikoja voidaan vertailla myös merkkijonoina, joita JSON tukee. JSON:in mahdollisia datatyyppejä ovat totuusarvomuuttuja (boolean), numero (number), merkkijono (string), tyhjä (null), olio (object) tai taulukko (array), eli JSON ei voi tallentaa suoraan dataa Date-tyyppisenä [50].

Kun aikadataa siirretään sovelluksen ja ohjelmointirajapinnan välillä, on tehtävä datan muunnosta merkkijonosta päiväykseksi ja päinvastoin. Suositus on, että

backendissä käytetään ISO 8601- ja UTC-aikoja ja ajat muunnetaan tarvittaessa paikallisiksi frontendissä. [45.]

4 Lomasovelluksen alustava toteutus

4.1 Backend ja API

Backendissa on käytetty paikallisessa kehityksessä MongoDB-tietokantaa ja mongoosea. Esimerkkikoodissa 4 näkyy mongoosen vacationer-skeema, joka kuvaa lomalainen-mallin, jonka mukaan tietokantaan tallennetaan työntekijät. Malli koostuu nimestä sekä taulukosta loma-aikoja, joilla on alku- ja loppuajankohta sekä mahdollinen kommentti-merkkijono. Lisäksi skeemassa määritellään name-kenttä pakolliseksi (required) sekä yksikäsitteiseksi (unique) eli kahdella oliolla ei voi olla samaa name-kentän arvoa.

```
const vacationerSchema = new mongoose.Schema({
    name: {    type: String,
               minlength: 3,
               required: true,
               unique: true},
    vacations: [
        {      comment: String,
               start: Date,
               end: Date}
    ]
})
```

Esimerkkikoodi 4. Sovelluksen vacationer-tietomallin Mongoose-skeema.

Sovelluksessa on toteutettu REST-ohjelmointirajapinta, jonka kuvaus näkyy taulukossa 2.

Taulukko 2. REST-rajapinnan kuvaus.

Osoite	**Metodi**	**Parametrit**	**Kuvaus**
/vacationers	GET		Palauttaa kaikki lomalaiset lomineen.
/vacationers	POST		Lisää uuden lomalaisen.
/vacationers/:id	GET		Palauttaa lomalaisen id-tunnisteella.
/vacationers/:id	DELETE		Poistaa lomalaisen id-tunnisteella.
/vacationers/:id	PUT		Päivittää lomalaisen id-tunnisteella.
/vacationeramount	GET	start, end	Hakee lomalaisten määrän aikavälillä start ja end (ISO 8601 standardi).
/timespan	GET	start, end	Hakee yksittäiset päivämäärät ja päivämäärien lomalaisten lukumäärän aikavälillä start ja end (ISO 8601 standardi).

Lomalla olevien määrää voidaan hakea eri tavoilla. Jos haetaan esimerkiksi ensi viikon lomalaisten määrää ja vastaus on ”ensi viikolla on 10 työntekijää lomalla”, ei vielä tiedetä lomien todellista vaikutusta. Montako työntekijää lomailee ensi perjantaina? Lomalaisten määrä tulisi laskea siis ainakin kahdella tavalla: ajanjakson kokonaismääränä sekä ajanjakson yksittäisten päivien määränä.

Ajanjakson yksittäisten päivien määrää varten haetaan aluksi tietokannasta kaikki lomat, joiden ajankohta on start- ja end-päivämäärien välissä eli loman yksikin päivä osuu päivämäärävälille (esimerkkikoodi 5). Tässä hyödynnetään mongoDB:n koostamisputkea (aggregation pipeline). Unwind-vaihe purkaa dokumentit vacations-tietokentän mukaan, ja match-vaihe valitsee lomat, joiden alku on ennen valitun ajanjakson loppua ja loppu on valitun ajanjakson alun jälkeen eli joiden ajat limittyvät valitun ajanjakson kanssa.

```
async function fetchVacationData(start, end) {
    const all = await Vacationer.aggregate([
        {     $unwind: {path: "$vacations"}},
        {     $match: {
                    $and: [{  "vacations.end": {
                                $gte: (new Date(start))}},
                           {"vacations.start": {
                                $lte: (new Date(end))}}]},
        }
     ])
    return all;}
```

Esimerkkikoodi 5. API:n timespan-pääteosoitteen hakufunktio.

Näitä päivämäärävälille osuvia lomia käytetään itse tapahtumankäsittelijässä, joka koostaa yksittäisten päivien lomailijamäärät (esimerkkikoodi 6). Käsittelijä hoitaa pyynnöt, jotka on lähetetty timespan-pääteosoitteeseen. API-osoite palauttaa taulukon kaksialkioisena: ensimmäinen alkio on päivämäärä ja toinen on lomailijoiden määrä päivämääränä.

```
async function handleVacationData(start, end) {
    const vacData = await fetchVacationData(start, end)
    let holidayTimes = []
    for (let i = 0; i < vacData.length; i++) {
        let vacationObj = {}
        vacationObj["start"] = new Date(vacData[i].vacations.start)
        vacationObj["end"] = new Date(vacData[i].vacations.end)
        holidayTimes.push(vacationObj)
     }
    let earlyDate = new Date(start)
    let lateDate = new Date(end)
    let arrayOfDates = []
    while (earlyDate <= lateDate) {
        let count = 0;
        holidayTimes.forEach(
            function (range) {
                if (earlyDate>=range.start && earlyDate<=range.end) {
                    count++}})
        let dateObject = []
        dateObject[0]= new Date(JSON.parse(JSON.stringify(earlyDate)))
        dateObject[1] = count
        arrayOfDates.push(dateObject)
        earlyDate.setDate(earlyDate.getDate() + 1)}
    return arrayOfDates;
}
```

Esimerkkikoodi 6. API:n timespan-pääteosoitteen tapahtumankäsittelijä, joka palauttaa päivämäärävälillä (start – end) kaikki yksittäiset päivämäärät ja lomailijoiden määrän kyseisenä päivämääränä.

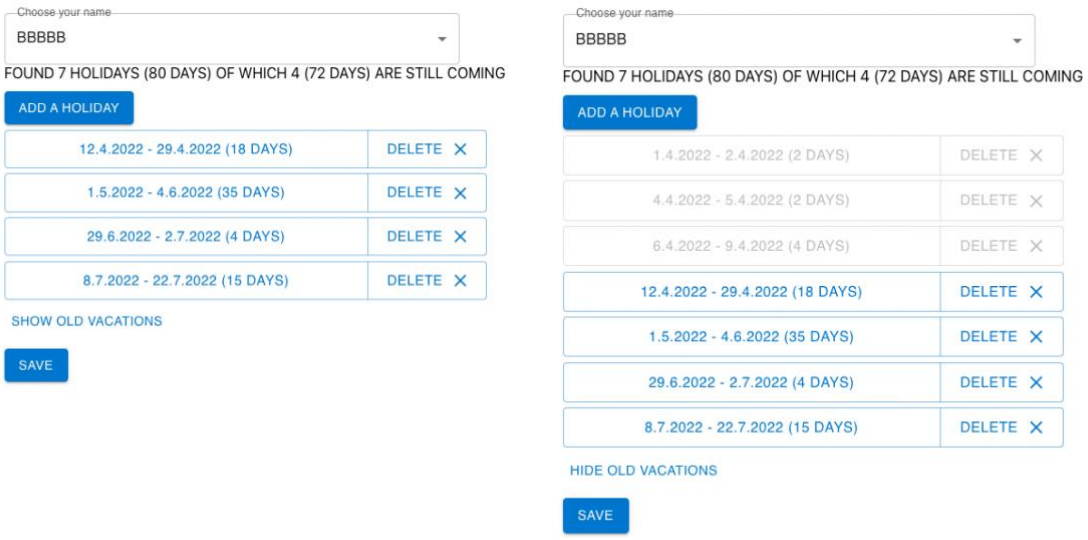
Tätä taulukkoa voidaan hyödyntää Slack-viesteissä sekä lomien muokkauslomakkeen varoitusasetuksissa. Taulukon avulla voidaan laskea myös lomalaisten määrä ajanjaksolla päivittäisenä keskiarvona, joka kertoo enemmän lomatilanteesta kuin pelkkä ajanjakson lomalaisten yhteismäärä.

4.2 Frontend ja UI-näkymät

Frontend jakaantuu kolmeen osaan, CRUD-lomakkeeseen, yleisnäkymään ja ylläpitonäkymään. Lomien CRUD-lomakkeella voidaan nähdä, lisätä, muokata ja poistaa tietokannassa olevien käyttäjien loma-aikoja. Lomakkeeseen liittyy myös ponnahdusikkuna, jossa on kalenterinäkymä päivämäärien valitsemista varten. Lomien yleisnäkymä on lomien jakaantumisen valvontaa varten. Näkymässä kuvataan kaikkien työntekijöiden lomien jakaantuminen valitun kuukauden ajalta. Ylläpitosivu on tarkoitettu ylläpitäjälle ja siellä voidaan lisätä ja poistaa käyttäjiä sekä muokata raja-arvoja, kuten lomalaisten maksimiarvoa tai työntekijöiden kokonaismäärää.

4.2.1 Lomien CRUD-lomake

CRUD-lomakkeella (kuva 6) voidaan nimensä mukaisesti suorittaa lomien CRUD-operaatioita eli lisätä, lukea, päivittää ja poistaa lomia. Lomakkeella on aina valittava pudotusvalikosta käyttäjä, koska sivustoon ei rakenneta kirjautumismahdollisuutta ainakaan toistaiseksi. Loma-ajoista näytetään oletusarvoisesti vain ne, jotka eivät ole vielä päättyneet. Käyttäjä voi suodattaa myös menneet lomat näkyviin halutessaan, mutta menneitä lomia ei voi muokata tai poistaa.



Kuva 6. Lomien CRUD-lomakkeen näkymän kaksi tilannetta, joissa vasemmalla menneet lomat (ennen käyttöpäivää päättyneet) on piilotettu (hide old vacations) ja oikealla ne ovat näkyvissä (show old vacations).

Loma-aikojen muokkaukseen ja uuden loman lisäämiseen käytetään ponnahdusikkunassa näkyvää kalenterinäkymää. Loman lisäys onnistuu painamalla add a holiday -painikkeesta, muokkaus loman ajankohdan painikkeesta ja poisto delete-painikkeesta.

4.2.2 CRUD-lomakkeen kalenterinäkymä

Lomien lisäämiseen ja muokkaamiseen käytetään ponnahdusikkunassa avautuvaa kalenterinäkymää, johon merkataan loman alku- ja loppupäivämäärä. Kalenteriksi vertailtiin eri avoimen lähdekoodin vaihtoehtoja, joka tukisi Reactia ja eri selaimia. Yksinkertaisuutensa, suosionsa (yli miljoona viikkolatausta npm:stä alkuvuonna 2022) ja selkeytensä vuoksi valittiin React Date Picker, jolla on avoin MIT-lisenssi [51], eli sitä voidaan käyttää myös kaupallisessa ohjelmistossa ja sen koodia voidaan myös muokata. MIT-

lisenssillisen ohjelmiston käyttöä varten on lisättävä lisenssiteksti mukaan lomasovelluksen lähdekoodiin. [52.]

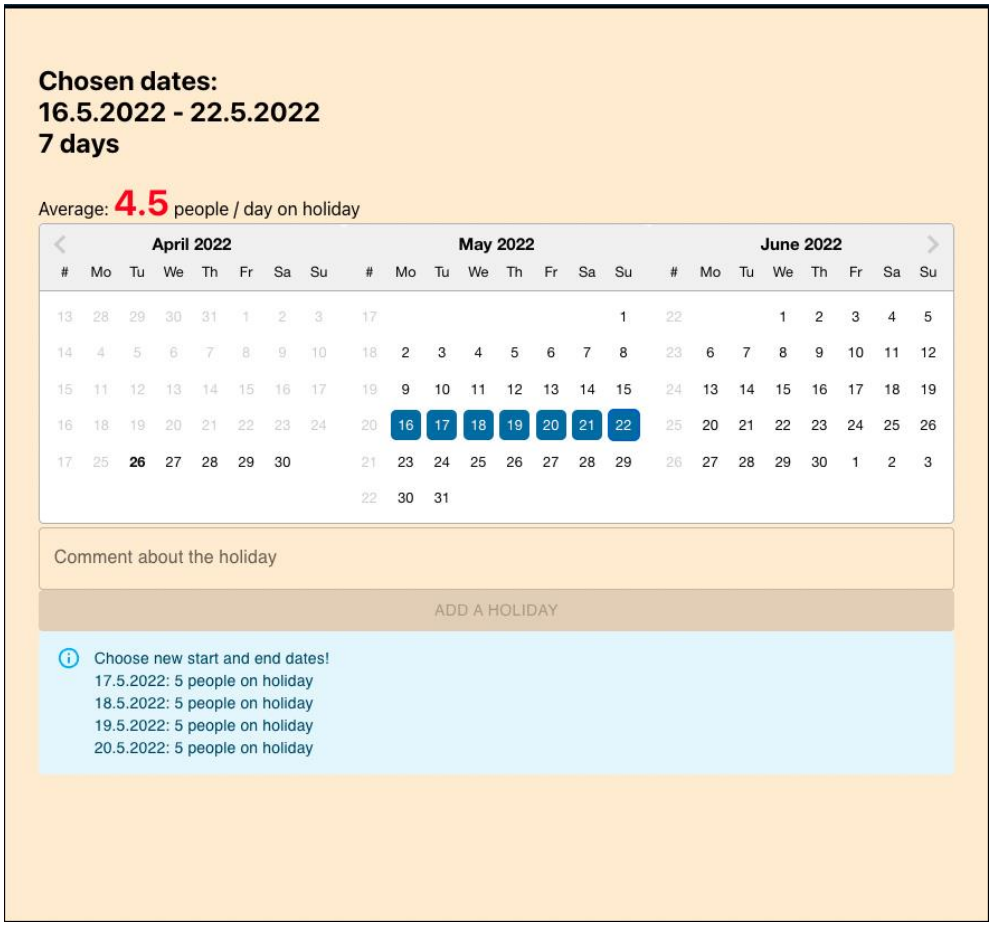
Esimerkkikoodissa 7 on Date Picker -kalenterikomponentti. Tämän kalenterin hyviä puolia ovat esimerkiksi mahdollisuus lisätä sekä aloitus- että päättymispäivä samasta kalenterista (selectsRange), lisätä poissuljettuja päivämääriä (excludeDateIntervals), käyttää lokalisointia (locale), muokata viikon aloituspäivää ja näytettävien kuukausien määrää (calendarStartDay ja monthsShown) sekä merkata väreillä päivämääräryhmiä CSS-tunnisteilla (highlightDates). Myös lomia lisätessä käyttäjän jo valitsemien loma-aikojen uudelleen valinta voidaan estää (calendarDaysExcluded).

```
<DatePicker
     locale="en"
     selected={startDate}
     onChange={onChange}
     selectsRange
     excludeDateIntervals={calendarDaysExcluded}
     startDate={startDate}
     endDate={endDate}
     minDate={!changingStartedSpace && today}
     dateFormat="dd.MM.yyyy"
     calendarStartDay={1}
     showWeekNumbers
     disabledKeyboardNavigation
     inline
     monthsShown={3}
     highlightDates={dayIsFullErrorMessage && alertingDates.map(a => {
          return a[0]
     })}
/>
```

Esimerkkikoodi 7. React Date Picker -elementti eli kalenterinäkymä, josta valitaan loman aloitus- ja lopetusaika (startDate ja endDate).

Kun päivämäärät on valittu kalenterinäkymästä (kuva 7), lähetetään API-pyyntö valituilla päivämäärillä timespan-osoitteeseen ja palautuneesta taulukosta lasketaan keskiarvo lomalaisten määrälle sekä tarkistetaan, ylittääkö yksittäisen päivän lomalaisten määrä raja-arvon. Esimerkiksi 10 hengen tiimissä voidaan määritellä lomalla olijoiden enimmäismäärän raja-arvoksi 5 yksittäisenä päivänä. Näin käyttäjä voi saada varoituksen, jos valitulla ajanjaksolla on päiviä,

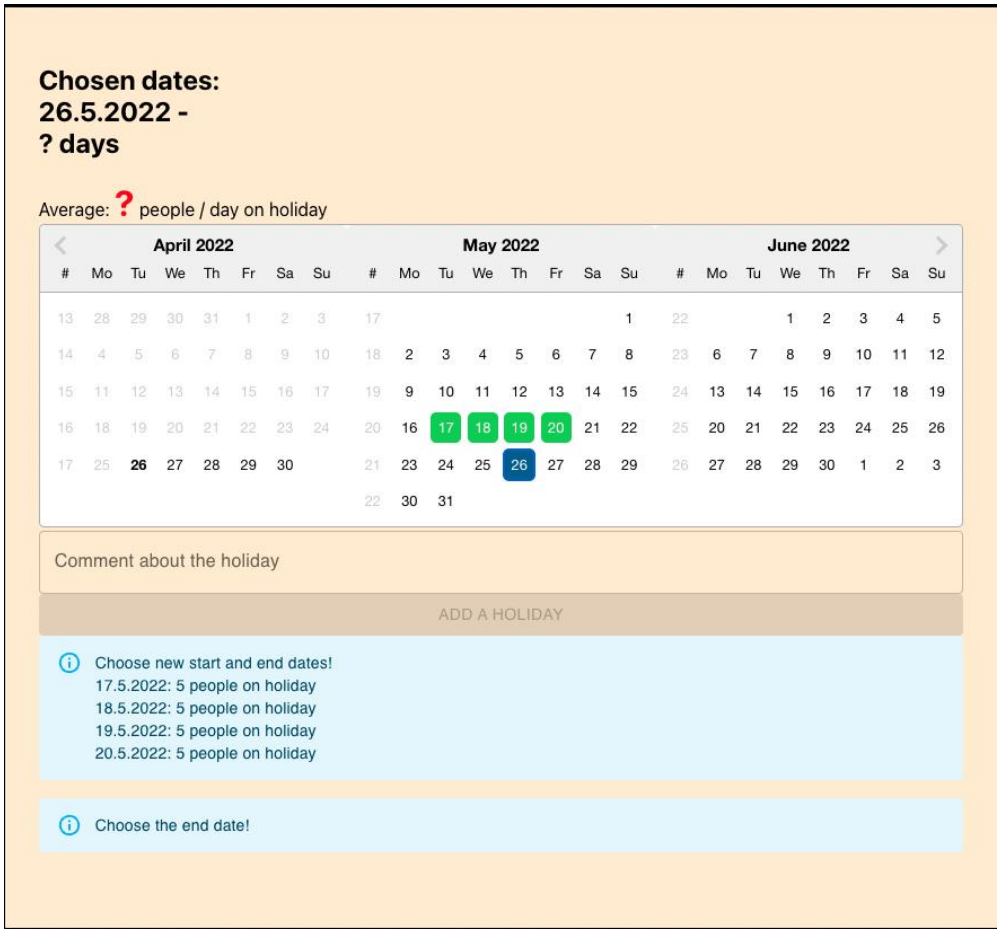
joita ei voi valita loma-ajaksi.



Kuva 7. CRUD-lomakkeen kalenterinäkymä: valitut päivät (sinisellä pohjalla), keskiarvo valittuina päivinä (punaisella) ja varoitus lomalaisten raja-arvon ylittävistä päivämääristä (vaaleansinisellä pohjalla).

Ennen kuin käyttäjä on valinnut päivämäärät uudelleen, nämä poissuljetut päivät on merkattu kalenteriin vihreällä (kuva 8). Tämä on toteutettu React Date Pickerin highlightDates-muuttujalla. Kun käyttäjä valitsee päivämäärät

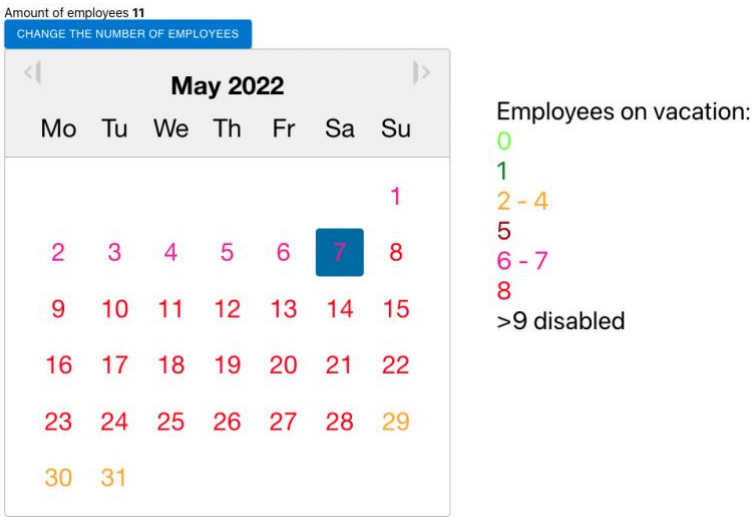
uudelleen, sovellus päivittää myös poissuljetut päivät.



Kuva 8. CRUD-lomakkeen kalenterinäkymä: raja-arvon (5) ylittäneet yksittäiset päivät korostettuna kalenterissa (vihreällä pohjalla).

4.2.3 Yleisnäkymä

Yleisnäkymän ensimmäinen koodattu versio perustui anonyymiyteen eli yleinen lomatilanne pyrittiin esittämään niin, ettei yksittäisiä työntekijöitä ja heidän loma-aikojaan voisi nähdä. Ratkaisussa hyödynnettiin uudelleen React Date Pickeriä. Valitun ajanjakson yksittäisen päivät merkattaisiin värikoodattuna siten, että värit edustaisivat tiettyä lomalaisten lukumäärää kyseisenä päivänä. Eri värikategoriat vaihtuisivat sen mukaan, mikä valittu työntekijöiden lukumäärä on. Yleisnäkymän prototyyppi näkyy kuvassa 9.



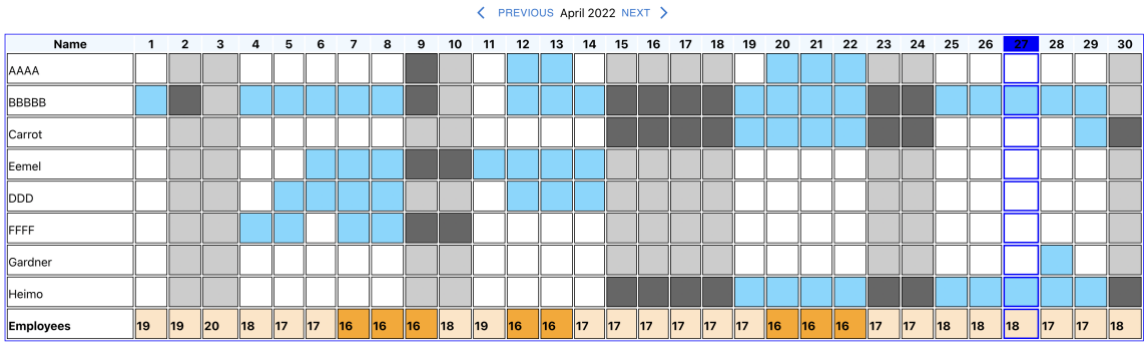
Kuva 9. Prototyyppi yleisnäkymän lämpökarttaversiosta.

Tämän lämpökarttakalenterin ongelmat ovat ilmeisiä: vaikka lämpökartta näyttää eri ajanjaksojen suosion lomien suhteen, juuri muuta informaatiota näkymä ei tarjoa. Yleisnäkymän tulisi esittää loma-ajat työntekijät eritellen. Muuten sijaisia ei voida valita tai loma-aikoja uudelleen limittää. Lisäksi värikoodattu kalenteri on epäselvä tapa esittää asia, eivätkä värisokeat voi käyttää sitä.

Yleisnäkymästä oli siis tehtävä uusi versio. Seuraavan version suunnittelussa pohdittiin nykyisen ratkaisun eli Excel-taulukoiden hyviä puolia. Excelistä näkee yhdellä näkymällä tietyllä ajanjaksolla kaikkien työntekijöiden lomatilanteen melko selkeästi. Yleisnäkymää voitaisiin rakentaa pohjaltaan käytössä olevan Excel-taulukon näköiseksi, jolloin siihen voitaisiin merkata päivämäärät viikonloppuineen ja yleisine juhlapäivineen sarakkeittain ja työntekijät riveittäin. Tällainen näkymä myös muistutti joitain kaupallisia ratkaisuja, joita lomasuunnitteluun on jo olemassa eli kyseinen ratkaisu on todettu toimivaksi vastaavissa sovelluksissa.

Lopullisessa yleisnäkymässä (kuva 10) voi valita kuukauden, jonka lomatilanne halutaan nähdä. Näkymä kokoaa yhteen valitun kuukauden kaikki päivät ja merkkaa riveittäin lomalaiset ja näiden yksittäiset loma-ajat valitun kuukauden ajalta. Yleisnäkymä myös laskee yhteen paikalla olevat työntekijät sarakkeittain

eli päivämäärien suhteen. Yleisnäkymää voidaan käyttää joko nimisarakkeen kanssa tai ilman sitä, jolloin näkymästä voidaan tehdä anonyymi tarvittaessa. Yleisnäkymässä on merkattuna lomat (vaaleansininen), yleiset vapaapäivät (vaaleanharmaa) sekä yleisten vapaapäivien aikaiset lomat (tummanharmaa). Yleisnäkymässä näkyvät myös tämänhetkinen päivämäärä (sininen) sekä päivittäin paikalla olevat työntekijät (oranssilla ja beigellä pohjalla) ja työntekijämäärän raja-arvon (17) alittaneet päivät (oranssi).



Kuva 10. Värikoodattu yleisnäkymä.

Yleisnäkymä on rakennettu React Table -hookeja hyödyntäen [53]. React Tablella on MIT-lisenssi, ja se on headless-ohjelmisto, eli sillä ei ole ennalta määriteltyä käyttöliittymänäkymää, vaan frontend voidaan ohjelmoida vapaasti käyttäen React Tablen hookeja, kuten useTablea. Yleisnäkymään on luotu myös virallisten suomalaisten juhlapäivien haku ilmaisesta ja avoimesta Nager.date-rajapinnasta, joka tarjoaa eri maiden juhlapäivädataa [54]. Taulukko merkkaa viikonloput ja juhlapäivät yhdellä värillä, lomapäivät toisella värillä ja näiden yhdistelmän kolmannella värillä. Taulukon kaikkia merkkausvärejä voidaan myös muokata ja käyttää lomien merkkaamiseen värin sijaan tekstiä tai symboleja, mitkä parantavat sovelluksen saavutettavuutta esimerkiksi värisokeutta ajatellen (kuva 11). Lisäksi tämä antaa käyttäjille mahdollisuuden vaihtaa kalenterin väriteemaa omien toiveidensa mukaiseksi.

< PREVIOUS April 2022 NEXT >

Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
AAAA									x			x	x							x	x	x								
BBBBB	x	x		x	x	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Carrot															x	x	x	x	x	x	x	x	x	x					x	x
Eemel						x	x	x	x	x	x	x	x	x																
DDD					x	x	x	x				x	x	x																
FFFF				x	x		x	x	x	x																				
Gardner																												x		
Heimo															x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Employees	19	19	20	18	17	17	16	16	16	18	19	16	16	17	17	17	17	17	17	16	16	16	17	17	18	18	18	17	17	18

Kuva 11. Yleisnäkymä, jonka lomat on merkattu X-kirjaimella värien sijaan.

4.2.4 Ylläpitonäkymä

Ylläpitonäkymässä voidaan lisätä ja poistaa käyttäjiä sekä säätää asetuksia, kuten työntekijöiden kokonaismäärää, joita käytetään yleisnäkymässä sekä lomailijoiden enimmäismäärän raja-arvoa, jota käytetään yleisnäkymässä ja CRUD-lomakkeen kalenterinäkymässä päivämäärien valinnassa. Ylläpitonäkymän kehitys on vielä kesken ja sitä kehitetään tilaajan toiveiden mukaan.

4.3 Slack-integraatio

Slack-ilmoitus toteutetaan cron jobin avulla myöhemmin, kun sovellus on saatu siirrettyä Azuren pilvipalvelimelle. Slack-sovellukseen lähetetään viikoittain automaattisena viestinä työryhmän kanavalle tieto lomalaisten määrästä seuraavana viikkona. Tämä voidaan toteuttaa viikoittaisella cron jobilla, joka lähettää sovelluksen API:in pyynnöt lomalaisten yhteismäärästä sekä yksittäisten päivämäärien lomalaisten määristä ja lähettää API:sta saadut tiedot Slackin webhookiin (esimerkkikoodi 8).

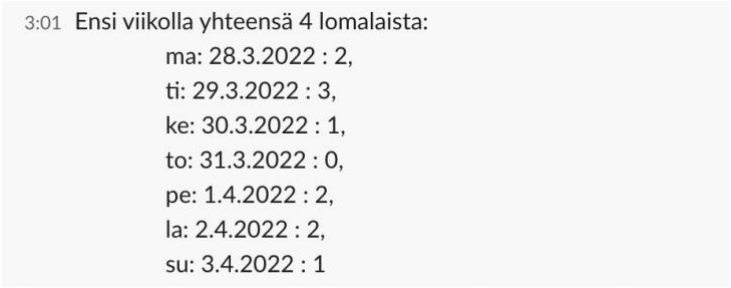
```
const cron = require("node-cron")
cron.schedule("0 12 * * 1", function (){
     let numberOfVacationers = 0;
     let vacationersPerDay = []

     axios.get(`http://<SERVER_ADDRESS>/vacationeramount?start=${nextMo
     nday.toISOString()}&end=${nextSunday.toISOString()}`)
          .then((response) => {
            numberOfVacationers = response.data.length;

          axios.get(`http://<SERVER_ADDRESS>/timespan?start=${nextMonda
          y.toISOString()}&end=${nextSunday.toISOString()}`)
               .then((response) => {
             vacationersPerDay = response.data;
                })
          .then(() =>
               axios.post(<SLACK_WEBHOOK_ADDRESS>, JSON.stringify({
     "text":
`Ensi viikolla yhteensä ${numberOfVacationers} lomalaista:
ma: ${new Date(vacationersPerDay[0][0]).toLocaleDateString()} :
${vacationersPerDay[0][1]},
ti: ${new Date(vacationersPerDay[1][0]).toLocaleDateString()} :
${vacationersPerDay[1][1]},
ke: ${new Date(vacationersPerDay[2][0]).toLocaleDateString()} :
${vacationersPerDay[2][1]},
to: ${new Date(vacationersPerDay[3][0]).toLocaleDateString()} :
${vacationersPerDay[3][1]},
pe: ${new Date(vacationersPerDay[4][0]).toLocaleDateString()} :
${vacationersPerDay[4][1]},
la: ${new Date(vacationersPerDay[5][0]).toLocaleDateString()} :
${vacationersPerDay[5][1]},
su: ${new Date(vacationersPerDay[6][0]).toLocaleDateString()} :
${vacationersPerDay[6][1]}`
     }))
})
...
```

Esimerkkikoodi 8. Luonnos viikoittain (kello 12 maanantaisin) toistuvasta Node.js-palvelimen cron jobista, joka lähettää viestin seuraavan viikon lomalaisten määristä tiimin Slack-ryhmään.

Kuvassa 12 näkyy esimerkki Slack-viestin muodosta, jossa kuvataan lomailevien työntekijöiden yhteismäärä sekä jakaantuminen viikon eri päiville. Näistä Slack-kanavaan automaattisesti tulevista viesteistä työryhmä saa tulevan viikon tilanteesta yleiskuvan, jonka avulla esimerkiksi työnsuunnittelu ja ennakointi helpottuvat. Viestiin voitaisiin jatkokehityksessä lisätä myös tieto lomailevien henkilöiden nimistä.



Kuva 12. Esimerkki työryhmän kanavaan lähetettävästä automaattisesta Slack-viestistä.

4.4 Aikamäärittelyt

Sovelluksessa käytetään UTC-aikoja. Kaikki API-pyyntöihin liittyvät aikamuuttujat määritellään JavaScriptin Date-objektin metodilla setUTCHours kellonajoiltaan 12:ksi. Loman alkamiskellonajaksi määritellään 10:00 ja loppumiskellonajaksi 14:00 backendin logiikan vuoksi. Kuvassa 13 näkyy lokitietoesimerkki, jossa timespan-pääteosoitteessa API tarkistaa kaikki lomat valitulla aikavälillä (1.-3.4.2022) päivä kerrallaan ja laskee, montako henkilöä on kyseisenä päivänä lomalla.

```
[0] onko  2022-04-01T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-02T14:00:00.000Z ?  true
[0] onko  2022-04-01T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-08T14:00:00.000Z ?  true
[0] Objekti [ 2022-04-01T12:00:00.000Z, 2 ]
[0] onko  2022-04-02T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-02T14:00:00.000Z ?  true
[0] onko  2022-04-02T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-08T14:00:00.000Z ?  true
[0] Objekti [ 2022-04-02T12:00:00.000Z, 2 ]
[0] onko  2022-04-03T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-02T14:00:00.000Z ?  false
[0] onko  2022-04-03T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-08T14:00:00.000Z ?  true
[0] Objekti [ 2022-04-03T12:00:00.000Z, 1 ]
[0] GET /timespan?start=2022-04-01T12:00:00.000Z&end=2022-04-03T12:00:00.000Z 200 48.159 ms - 94
```

Kuva 13. Backendin lokia GET-kutsusta timespan-pääteosoitteeseen.

Algoritmi vertaa valitun aikavälin päiviä kaikkiin loma-aikoihin ja merkkaa boolean-totuusarvolla, onko kyseinen päivämäärä osa loma-aikaa. GET-kutsussa käytetään UTC-kellonaikaa 12:00, jotta varmistetaan kellonaikojen

limittyminen. Kuvan 13 lokissa näkyy myös palautuvan taulukon alkiot (Objekti), joissa on päivämäärä sekä lomalaisten määrä päivämääränä.

Syy näihin eriäviin lomien alku- ja loppuaikoihin on kesäaikaongelma, joka näkyy kuvassa 14. Tilanteessa GET-kutsun päivämäärät timespan-pääteosoitteeseen sijoittuvat kesäajan vaihtumisen molemmille puolille. API-pyyntö käyttää UTC-kellonaikaa 12, mutta kesäajan vaihtuessa (27.3.2022) pyyntö vaihtuu kellonajaksi 11. Mikäli lomat alkaisivat ja päättyisivät kello 12, limittymistä ei tapahtuisi eikä backend toimisi oikein. Kun loma-ajat on määritelty alkamaan kello 10 ja päättymään kello 14, kesäaika ei aiheuta ongelmaa. Käyttämällä yhdenmukaisesti UTC-aikoja voidaan varmistaa, että päivät limittyvät riittävällä tasolla sovelluksen toimimiseksi.

```
[0] onko  2022-03-26T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-02T14:00:00.000Z ?  false
[0] onko  2022-03-26T12:00:00.000Z  aikavälillä  2022-03-25T10:00:00.000Z  -  2022-03-29T14:00:00.000Z ?  true
[0] onko  2022-03-26T12:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-08T14:00:00.000Z ?  false
[0] onko  2022-03-26T12:00:00.000Z  aikavälillä  2022-03-29T10:00:00.000Z  -  2022-03-30T14:00:00.000Z ?  false
[0] onko  2022-03-26T12:00:00.000Z  aikavälillä  2022-03-25T10:00:00.000Z  -  2022-03-27T14:00:00.000Z ?  true
[0] Objekti [ 2022-03-26T12:00:00.000Z, 2 ]
[0] onko  2022-03-27T11:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-02T14:00:00.000Z ?  false
[0] onko  2022-03-27T11:00:00.000Z  aikavälillä  2022-03-25T10:00:00.000Z  -  2022-03-29T14:00:00.000Z ?  true
[0] onko  2022-03-27T11:00:00.000Z  aikavälillä  2022-04-01T10:00:00.000Z  -  2022-04-08T14:00:00.000Z ?  false
[0] onko  2022-03-27T11:00:00.000Z  aikavälillä  2022-03-29T10:00:00.000Z  -  2022-03-30T14:00:00.000Z ?  false
[0] onko  2022-03-27T11:00:00.000Z  aikavälillä  2022-03-25T10:00:00.000Z  -  2022-03-27T14:00:00.000Z ?  true
[0] Objekti [ 2022-03-27T11:00:00.000Z, 2 ]
```

Kuva 14. Kesäajan ongelma GET-kutsuissa timespan-osoitteeseen.

4.5 Jatkokehitys

Yksi projektin tavoitteista oli saada loman lisäys helpoksi. Tällä hetkellä loman lisätäkseen käyttäjän on tehtävä useampi vaihe: valita käyttäjä, avata kalenterinäkymä, valita päivämäärät ja tallentaa loma kahdella painikkeella. Lomien yleisnäkymä on selkeä, ja sen päälle voisi kehittää kaikki lomien CRUD-toiminnallisuudet lukemisen lisäksi eli lomien lisäyksen, poiston ja muokkauksen, mikä tekisi CRUD-toiminnallisuudet nopeammiksi ja helpommiksi. Tämä muutos vaatisi kuitenkin koodiin isoja muutoksia.

Sovellus vaatii pieniä viimeistelyjä, kuten lopullisen Slack-viestin sisällön ja näkymien tyylien valinnan. Sovellusta on myös yhä arvioitava tilaajan kanssa

nykytilanteen ja jatkokehityksen osalta. Sovellus voisi muistuttaa työntekijöitä loma-aikojen ilmoittamisesta tai antaa työnjohdolle mahdollisuuden hyväksyä lomia. Sovellusta voitaisiin kehittää ottamaan huomioon myös lomien ansaitseminen. Se voisi pitää kirjaa ansaituista ja käytetyistä lomista lomakausien mukaisesti ja auttaa yhä paremmin lomien suunnittelussa.

Kun sovellus on hyväksytty tilaajan puolesta valmiiksi, se siirretään pilvipalveluun ja siihen lisätään suunniteltu Slack-integraatio. Tämän jälkeen voidaan aloittaa käyttäjätestaus tilaajayrityksen työryhmässä.

5 Yhteenveto

Insinöörityössä pyrittiin määrittelemään, suunnittelemaan ja toteuttamaan alustava versio lomasovelluksesta tilaajayrityksen käyttöön sekä valitsemaan web-teknologiapino sovellukselle. Lomasovelluksella voisi tarkastella, lisätä, muokata ja poistaa lomakausia työryhmän työntekijöille sekä seuraamaan lomakausien jakaantumista eri ajanjaksoilla. Web-teknologiaksi vertailtiin MEAN-, MERN-, MEVN-, LAMP-, LEMP-, WISA-, SMACK-, JAMStack-, Ruby on Rails- ja Django-pinoja sekä Spring-kehykseen perustuvaa ratkaisua.

Tiedonhakuvaiheessa huomattiin, kuinka eri pinojen suosio vaihtelee ja niillä on omat sopivat käyttöympäristönsä. On myös huomattava, että tarkastellut pinovaihtoehdot ovat suurimmaksi osaksi kokonaisratkaisuja, mutta pinon teknologioita voidaan myös korvata vaihtoehtoisilla teknologioilla. Tässä tarkastelussa pyrittiin keskittymään suosittuihin pinovaihtoehtoihin, mutta esitettyjen pinojen ulkopuolisia ja soveltuvia vaihtoehtoja on lukuisia.

Sovelluksessa käytettäväksi pinoksi valittiin MERN eli MongoDB-tietokanta, Node/Express-backend ja React-kirjasto. Ohjelmointirajapinnaksi vertailtiin eri API-protokollia, joista valittiin REST sovelluksen yksinkertaisten backend-vaatimusten vuoksi. REST-rajapinta on suunniteltu tukemaan integraatioita, kuten automaattisten Slack-ilmoitusten käyttöä.

Sovelluksessa voi tällä hetkellä tehdä lomien CRUD-operaatioita, tarkastella lomien yleistä jakaantumista kuukausittain ja säätää yleisiä asetuksia. Suurin osa projektin alussa luoduista käyttötilanteista onnistuu sovelluksessa. Toisaalta sovellukseen voidaan jatkossa lisätä määrittelyssä mainittuja ja tällä hetkellä puuttuvia ominaisuuksia. Kun sovellus on saatu ominaisuuksiltaan valmiiksi, se viedään Azuren pilvipalveluun, minkä jälkeen jo hahmoteltu Slack-integraatio saadaan lisättyä ja käyttäjätestaus aloitettua.

Yleisnäkymän luomisessa keskityttiin aluksi etenkin yksityisyyden suojaan: yleisnäkymässä ei näytettäisi työntekijöiden nimiä suoraan. Tämän vuoksi ratkaisuksi haettiin värikoodattua kalenterinäkymää, joka piilottaisi yksittäisten työntekijöiden henkilöllisyydet. Tämä näkymä osoittautui kuitenkin sekavaksi ja saavutettavuudeltaan huonoksi ja yleisnäkymästä tehtiin uusi versio.

Lopullinen versio yleisnäkymän käyttöliittymälle pohjautui taulukkoon, joka on hyvin informatiivinen ja tuttu työryhmälle, koska se muistuttaa muodoltaan nykyään käytössä olevaa Excel-ratkaisua. Taulukko voidaan myös muokata saavutettavuudeltaan hyväksi. Taulukkomuodon huonoja puolia on yksittäisten työntekijöiden loma-aikojen näkyminen. Taulukkoon voitaisiin mahdollisesti jatkossa integroida myös lomakenäkymän toiminnallisuudet. Perinteinen CRUD-lomake päivämäärävalintoineen on vanhanaikainen, eikä ole käyttäjäkokemukseltaan vielä riittävän innostava. Jälkikäteen voidaan arvioida, että alkuvaiheen kehityksessä huomion olisi pitänyt keskittyä yleisnäkymään, ei loma-aikojen CRUD-toiminnallisuuteen. Sovellus olisi yksinkertaisempi ja selkeämpi, jos yleisnäkymä sisältäisi kaikki toiminnallisuudet.

Lomasovelluksen kehittäminen jatkuu myös insinöörityön jälkeen ja tavoitteena on saada sovellus aktiiviseen käyttöön työryhmässä lähitulevaisuudessa. Insinöörityö on koonnut tietoa eri web-teknologiapinoista, vahvistanut tekijän MERN-ohjelmointitaitoja ja päätynyt alustavaan versioon yrityksen lomasuunnittelusovelluksesta.

Lähteet

1 Tietoevry lyhyesti. 2022. Verkkoaineisto. Tietoevry. <https://www.tietoevry.com/fi/meista/tietoevry-lyhyesti>. Luettu 26.4.2022.

2 Usein kysyttyä EU:n tietosuoja-asetuksesta. 2022. Verkkoaineisto. Tietosuojavaltuutetun toimisto. <https://tietosuoja.fi/gdpr>. Luettu 2.3.2022.

3 Arun, Thomas. 2020. Top 6 tech stacks that reign software development in 2021. Verkkoaineisto. Fingent. <https://www.fingent.com/blog/top-6-tech-stacks-that-reign-software-development-in-2021>. Luettu 10.3.2022.

4 Marijan, Bosko. 2021. MEAN Vs. LAMP: Which is better? Verkkoaineisto. Phoenixnap. <https://phoenixnap.com/kb/mean-vs-lamp>. Luettu 10.3.2022.

5 What is NoSQL. 2022. Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/nosql-explained>. Luettu 17.4.2022.

6 Heike, Christoph. 2019. LAMP vs. MEAN: Which stack is right for you? Verkkoaineisto. Atlassian. <https://bitbucket.org/blog/lamp-vs-mean-which-stack-is-right-for-you>. Luettu 10.3.2022.

7 Kapoor, Ajay. 2021. Benefits of using MERN Stack. Verkkoaineisto. Medium. <https://enlear.academy/benefits-of-using-mern-stack-7e0c732b5214>. Luettu 12.4.2022.

8 March 2022 Web Server Survey. 2022. Netcraft. Verkkoaineisto. <https://news.netcraft.com/archives/2022/03/29/march-2022-web-server-survey.html>. Luettu 2.5.2022.

9 Stevens, Gary. 2022. Choosing Between Apache and NGINX for Your Web Hosting Needs. Hosting Canada. <https://hostingcanada.org/nginx-vs-apache-explained>. Luettu 28.4.2022.

10 Simic, Sofija. 2022. What is LAMP Stack? Verkkoaineisto. Phoenixnap. <https://phoenixnap.com/kb/what-is-a-lamp-stack>. Luettu 10.3.2022.

11 Rakov, Vladimir. 2019. How to Install the WISA (Windows, IIS, SQL, ASP.NET) Stack. Verkkoaineisto. HostAdvice. <https://hostadvice.com/how-to/how-to-install-the-wisa-windows-iis-sql-asp-net-stack>. Päivitetty 4.11.2019. Luettu 4.5.2022.

12 The Good and the Bad of .NET Framework Programming. 2022. Verkkoaineisto. Altexsoft. <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming>. Luettu 27.4.2022.

13 Estrada, Raúl. 2016. Fast Data Processing Systems with SMACK Stack. Packt Publishing.

14 A beginners guide to the SMACK stack – Parti 1: Introduction. 2016. Verkkoaineisto. <https://www.luminis.eu/blog/data-en/a-beginners-guide-to-the-smack-stack-part-1-introduction>. Luettu 10.4.2022.

15 What is JAMStack? 2022. Verkkoaineisto. Cloudflare, Inc. <https://www.cloudflare.com/en-gb/learning/performance/what-is-jamstack>. Luettu 18.4.2022.

16 Gienow, Michelle. 2017. The Sweetness of JAMStack: JavaScript, APIs and Markup. Verkkoaineisto. The New Stack. <https://thenewstack.io/the-sweetness-of-jamstack-javascript-apis-and-markup>. Luettu 18.4.2022.

17 Spring Framework – Overview. 2022. Verkkoaineisto. Tutorials Point. <https://www.tutorialspoint.com/spring/spring_overview.htm>. Luettu 18.4.2022.

18 Mulders, Michiel. 2019. What is Spring Boot? Verkkoaineisto. Stackify. <https://stackify.com/what-is-spring-boot>. Luettu 18.4.2022.

19 Rak Viktor. 2021. Pros and Cons of Ruby on Rails. Verkkoaineisto. Sloboda Studio. <https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails>. Luettu 2.5.2022.

20 Deery, Matthew. 2021. What Is the Django Framework? The Complete Beginner's Guide. Verkkoaineisto. CareerFoundry. <https://careerfoundry.com/en/blog/web-development/django-framework-guide>. Luettu 2.5.2022.

21 What is serverless computing? | Serverless definition 2022. Verkkoaineisto. Cloudflare, Inc. <https://www.cloudflare.com/en-gb/learning/serverless/what-is-serverless>. Luettu 10.3.2022.

22 Introduction to Azure Functions. 2022. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview>. Luettu 10.3.2022.

23 2021 Developer Survey. Verkkoaineisto. Stack Overflow. <https://insights.stackoverflow.com/survey/2021 >. Luettu 10.3.2022.

24 MERN Stack Explained. 2022. Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/mern-stack>. Luettu 2.3.2022.

25 How does the MERN stack work? 2020. Verkkoaineisto. Bocasay. <https://www.bocasay.com/how-does-the-mern-stack-work>. Luettu 13.4.2022.

26 Express/Node introduction. 2022. Verkkoaineisto. Mozilla Corporation. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction>. Luettu 6.3.2022.

27 Express middleware. Verkkoaineisto. OpenJS Foundation. <https://expressjs.com/en/resources/middleware.html>. Luettu 6.3.2022.

28 React. 2022. Verkkoaineisto. Meta Platforms, Inc. <https://reactjs.org>. Luettu 2.3.2022.

29 Tutorial: Intro to React. 2022. Verkkoaineisto. Meta Platforms, Inc. <https://reactjs.org/tutorial/tutorial.html>. Luettu 9.3.2022.

30 Introducing Hooks. 2022. Verkkoaineisto. Meta Platforms, Inc. <https://reactjs.org/docs/hooks-intro.html>. Luettu 17.4.2022.

31 Hooks at a Glance. 2022. Verkkoaineisto. Meta Platforms, Inc. <https://reactjs.org/docs/hooks-overview.html>. Luettu 17.4.2022.

32 Vetter-Neo, Natalia. 2020. When to use a UI component library in a React project? Verkkoaineisto. Sunscrapers. <https://sunscrapers.com/blog/when-to-use-a-ui-component-library-in-a-react-project>. Luettu 8.3.2022.

33 Advantages of MongoDB. 2021. Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/advantages-of-mongodb>. Luettu 6.3.2022.

34 Introduction to MongoDB. 2021. Verkkoaineisto. MongoDB, Inc. <https://docs.mongodb.com/manual/introduction>. Luettu 9.3.2022.

35 MongoDB Manual 5.0. 2021. Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/docs/manual>. Luettu 10.4.2022.

36 Kukic, Ado & Vlaeva, Stanimira. 2021. MongoDB & Mongoose: Compatibility and Comparison. Verkkoaineisto. MongoDB, Inc. <https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver>. Päivitetty 6.4.2022. Luettu 28.4.2022.

37 What is API: Definition, Types, Specifications, Documentation. 2021. Verkkoaineisto. Altexsoft. <https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation>. Luettu 18.3.2022.

38 What is gRPC: Concepts, Pros and Cons, Use Cases. 2021. Verkkoaineisto. Altexsoft. <https://www.altexsoft.com/blog/what-is-grpc>. Luettu 4.5.2022.

39 REST APIs. Verkkoaineisto. 2021. IBM. <https://www.ibm.com/cloud/learn/rest-apis>. Luettu 20.3.2022.

40 GraphQL is the better REST. Verkkoaineisto. How to GraphQL. <https://www.howtographql.com/basics/1-graphql-is-the-better-rest>. Luettu 18.3.2022.

41 Build. Verkkoaineisto. Slack. <https://slack.dev/guides/Build.pdf>. Luettu 2.3.2022.

42 Sending messages using Incoming Webhooks. 2022. Verkkoaineisto. Slack API. <https://api.slack.com/messaging/webhooks>. Luettu 2.3.2022.

43 Cron Job: A Comprehensive Guide for Beginners 2022. 2022. Verkkoaineisto. Hostinger Tutorials. <https://www.hostinger.com/tutorials/cron-job>. Luettu 17.4.2022.

44 Nwamba, Chris. 2019. How To Use node-cron to Run Scheduled Jobs in Node.js. Verkkoaineisto. DigitalOcean, LLC. <https://www.digitalocean.com/community/tutorials/nodejs-cron-jobs-by-examples>. Päivitetty 1.12.2021. Luettu 17.4.2022.

45 The Definitive Guide to DateTime Manipulation. Verkkoaineisto. Toptal. <https://www.toptal.com/software/definitive-guide-to-datetime-manipulation>. Luettu 17.3.2022.

46 Korpela, Jukka. 2013. Info on ISO 8601, the date and time representation standard. Verkkoaineisto. <https://jkorpela.fi/iso8601.html>. Luettu 15.3.2022.

47 Liew, Zell. 2019. Everything You Need To Know About Date in JavaScript. Verkkoaineisto. DigitalOcean. <https://css-tricks.com/everything-you-need-to-know-about-date-in-javascript>. Päivitetty 11.5.2020. Luettu 17.3.2022.

48 Project Status. Verkkoaineisto. Moment.js Documentation. <https://momentjs.com/docs/#/-project-status>. Luettu 17.3.2022.

49 Date. 2022. Verkkoaineisto. Mozilla Corporation. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date>. Luettu 15.3.2022.

50 Introducing JSON. Verkkoaineisto. <https://www.json.org/json-en.html>. Luettu 15.3.2022.

51 React Date Picker. 2022. Npm, Inc. Verkkoaineisto. <https://www.npmjs.com/package/react-datepicker>. Luettu 15.3.2022.

52 Open Source Software Licenses 101: The MIT License. 2021. FOSSA, Inc. Verkkoaineisto. <https://fossa.com/blog/open-source-licenses-101-mit-license>. Luettu 15.3.2022.

53 Linsley, Tanner, 2020. React Table. Verkkoaineisto. <https://react-table.tanstack.com>. Luettu 17.4.2022.

54 Nager.date Worldwide public holiday. Verkkoaineisto. 2022. Nager.at. <https://date.nager.at> Luettu 24.4.2022.