

Microsoft GRAPH-palvelun liittäminen SPA-sovellukseen



Ammattikorkeakoulututkinnon opinnäytetyö

Tieto ja viestintätekniikka, insinööri (AMK)

syksy 2022

Olli Rytkönen

Tieto ja viestintätekniikka

Tekijä Olli Rytönen

Työn nimi Microsoft Graph palvelun liittäminen SPA-sovellukseen

Ohjaaja Petri Kuittinen

Tiivistelmä

Vuosi 2022

Opinnäytetyön tavoite on liittää Microsoft Graph -palvelu SPA-sovellukseen. Sovellus on yksinkertainen treenisovellus, jossa hyödynnetään Microsoft Graph -palvelun kautta käyttäjän henkilökohtaisia tietoja. Käyttäjä voi tehdä sovelluksen kautta merkinnän Outlook kalenteriin. Tavoitteena on, että sovellus käyttää MongoDB-tietokantaa, josta tieto siirtyy Graph-palveluun.

Tekniikat-osiossa käydään kattavasti läpi sovelluksessa käytettävät tekniikat ja kilpailijat sekä niiden hyvät ja huonot puolet. Tekniikat-luvussa käydään läpi tietokannat, palvelin ja selain puolen sovelluskehikset, koodieditorit ja Graph-rajapinta.

Toteutus -luvussa käydään koodista tärkeimmät osiot läpi ja katsotaan, miten sovellus rekisteröidään käyttämään Microsoft Graph-rajapintaa. Luku sisältää koodi näytteitä ja kuvia valmiista sovelluksesta.

Johtopäätökset luvussa käydään läpi onnistumiset ja kehitettävät kohteet. Kehitettävää sovellukseen paljon ja alkuperäinen tavoite ei toteutunut. MongoDB-tietokannasta Graph-palveluun tiedon siirto ei onnistunut, joten sovellukseen tehtiin vain suora yhteys lisätä merkintä omaan Outlook-kalenteriin. Treenin aika ei myöskään siirtynyt oikein viallista aikavyöhykkeestä. Sovelluksen ulkoasuun myös jäi parannettavaa.

Avainsanat Office365, Microsoft Graph, React, Fullstack, SPA

Sivut 36 sivua

The thesis focuses in connecting Microsoft Graph service into SPA-application. The Application is a simple training app which uses Microsoft Graph service to access user's personal information. The user can mark training sessions to Outlook calendar. Goal is to transfer data from MongoDB database to Graph -service.

The techniques section covers widely techniques used in web-application and they rival techniques pros and cons. This part presents databases, back and front-end frameworks, code editors and Graph-service.

The execution paragraph reviews the most important features of code and undergoes how to register your app to use Microsoft Graph-service. The paragraph includes code examples and pictures from complete web-application.

The conclusion paragraph experience succeeds and improvement points. There is a lot of improvement in web-application and original goal did not reach. Transfer data between MongoDB's database and Microsoft Outlook calendar did not work so solution was to add training session straight to the Outlook calendar from the web-application. Training time also did not transferred correctly depending problems on wrong time-zone. Web-applications appearance also needs development.

Keywords M365, Microsoft Graph, ReactJS, Full stack developer, SPA

Pages 36 pages

Sisällysluettelo

| | |
|---|----|
| Käsitteistö | |
| 1 Johdanto | 1 |
| 2 Tekniikat | 1 |
| 2.1 Single Page App (SPA) | 1 |
| 2.2 Node.js & Express.js | 3 |
| 2.3 Tietokanta | 4 |
| 2.4 SQL-tietokanta | 5 |
| 2.5 Tietokantahallintajärjestelmä | 6 |
| 2.6 NoSQL-tietokanta..... | 7 |
| 2.7 Document database | 7 |
| 2.8 Key-Value database..... | 8 |
| 2.9 Column-oriented database | 8 |
| 2.10 Graph database | 9 |
| 2.11 Sovelluskehikset | 9 |
| 2.12 React.Js..... | 11 |
| 2.13 Microsoft Identity Platform | 11 |
| 2.14 Microsoft Graph | 13 |
| 2.15 Koodieditorit ja kehitysympäristöt | 15 |
| 3 Toteutus..... | 16 |
| 3.1 Nodejs/Express | 17 |
| 3.2 Node/Express sovelluksen runko | 18 |
| 3.3 REST-rajapinnan luominen..... | 19 |
| 3.4 MongoDB määrittely..... | 22 |
| 3.5 React sovelluksen toteuttaminen | 23 |
| 3.6 Sovelluksen määrittelyt | 24 |
| 3.7 Sovelluksen runko | 25 |
| 3.8 Muut komponentit..... | 27 |
| 3.9 Microsoft Graph | 29 |
| 4 Johtopäätökset | 34 |
| 5 Lähteet..... | 35 |

Käsitteistö

| | |
|-----------------------------|---|
| Azure AD B2C | Azure tunnistautumis- protokolla esimerkiksi Google ja Facebook tileille. |
| ADAL | Azure Active Directory Authentication Library. |
| OpenID Connect | OAuth 2.0 tunnistautumis- protokollan osa, joka kommunikoi tunnistautumis- serverin kanssa. |
| Microsoft Graph | Microsoftin palvelu käyttäjän Office 365- datan käsittelyyn. Osa Microsoftin tunnistautumis- alustaa. |
| Microsoft Identity Platform | Microsoftin tunnistautumisalusta. Pitää sisällään esimerkiksi Graph-palvelun, jolla päästään käsiksi käyttäjän tietoihin. |
| MSAL | Microsoft Authentication Libraries (MSAL) eli avoimen lähdekoodin tunnistautumiskirjasto. |
| WOD | Crossfit sanastoa, joka tarkoittaa päivän treeniä. Work Out of the Day. |

1 Johdanto

Opinnäytetyön idea muodostui työelämästä, jossa joudutaan käyttämään paljon erilaisia ohjelmia ja muistamaan paljon salasanoja. Työssä voidaan myös joutua tekemään samoja toistuvia toimintoja eri sovellusten välillä. Microsoft Graph -palvelua käyttämällä voidaan yhdistää sovelluksia kommunikoimaan Microsoftin pilvipalveluiden kanssa ja välttämään sovellusten välillä turhaa sovelluksien vaihtamista ja kirjautumista.

Opinnäytetyön tarkoituksena on luoda yksinkertainen web-sovellus, joka hyödyntää käyttäjän Microsoft-tilin tietoja. Sovellukseen tulee mahdollisuus luoda, päivittää ja poistaa treeni. Käyttäjä voi lisätä treenistä merkinnän itselleen Outlook-kalenteriin.

Opinnäytetyössä käydään läpi tärkeimmät tekniset ratkaisut ja niiden kilpailijat, myös niiden hyvät ja huonot puolet. Käyttöliittymän pitkälle kehittäminen ja koodin optimointi on rajattu pois työstä.

2 Tekniikat

2.1 Single Page App (SPA)

Single page app on JavaScript-sovelluskehyksellä toteutettu web-sovellus, joka lataa sovelluksen sisällön lähes kokonaan sovelluksen käynnistyessä. Perinteinen web-sivusto puolestaan on jaettu useaan sivuun. Sivusto latautuu uudelleen käyttäjän suorittaessa toiminnon tai siirtyessä sivulta toiselle. Monen sivun sovellus kommunikoi enemmän palvelimen kanssa. Yhden sivun sovelluksesta on tarkoitus tehdä nopea ja reaaliaikainen välttämällä sivun latausaikoja. Hybridi lähestyminen on myös mahdollista. Esimerkiksi laajempi moni sivuinen perinteinen web-sivu voi pitää sisällään yhden sivun sovelluksia.

Monisivuisen sovelluksen logiikka suoritetaan serverillä ja yhden sivun sovelluksen logiikka puolestaan selaimessa, joka kommunikoi ohjelmointi rajapinnan avulla web-serverille.

Yhden sivun sovellus koostuu yhdestä HTML-sivusta, joka renderöi kaiken sovelluksen näytettävän sisällön näytölle. Näytettävää sisältöä manipuloidaan JavaScript-koodilla. Yhden sivun

sovelluksessa käyttäjälle renderöidään vain erilainen näkymä sivun uudelleen lataamiseen sijaan. Käyttäjän liikkua tai tekemällä toimintoja sivulla päivitetään sivulla näytettäviä osia.

Ennen sovelluskehityksen aloittamista täytyy selvittää, kumpi on parempi tapa toteuttaa sovellus. Ensin on hyvä selvittää, mitä käyttäjä tekee sovelluksessa ja mikä on sovelluksen sisältö. Mitä käyttäjälle näytetään ja mitä käyttäjä voi konkreettisesti tehdä sovelluksessa? Onko sivusto interaktiivinen vai onko sivu tarkoitettu yhden suuntaiseen viestintään?

Microsoftin dokumentaatioissa perustella ("Choose Between Traditional Web Apps and Single Page Apps (SPAs)" 2022), että perinteisiä web-sovelluksia on hyvä käyttää, kun sovellus pysyy hyvin yksinkertaisena tai vain luku oikeudella varustettuna sivustona. Monen sivun sovelluksessa on etuna helpompi hakukoneoptimointi ja käyttäjä voi itse lisätä kirjanmerkkejä helposti.

Yhden sivun sovelluksen käyttämistä on syytä harkita, jos sovelluksessa on paljon toimintoja ja joudutaan käyttämään joka tapauksessa ohjelmointirajapintaa. Yhden sivun sovellus voi tehdä datan siirtämistä taustalla ja sivun toiminnot ovat reaaliaikaisia, koska sivun latauksia ei tarvitse suorittaa jokaisella toiminnolla. SPA-sovelluksiin voidaan rakentaa osoiteriville näkyvä osoite minkä voi lisätä kirjanmerkkeihin.

SPA-sovelluksen kehittämiseen sopii paikallinen selainympäristö, joka mahdollistaa ketterän sovelluksen kehittämisen. Sovellusta voi suorittaa selaimessa ja esimerkiksi Google Chrome -selaimen on mahdollista liittää lisäosia, jotka hyödyttävät kehitystä. Mitä nopeammin sovellusta pystyy suorittamaan, sitä nopeammin saadaan sovelluksen virheet nopeasti kiinni ja koodia korjattua. Mobiiliystävällisen yksisivuisesta sovelluksesta tekee mahdollisuus uudelleen käyttää palvelinpuolen koodia natiivi mobiilisovelluksessa. Yhden sivun sovelluksen tarkoitus on tuottaa mahdollisimman hyvä käyttäjäkokemus etenkin mobiilikäyttäjille. Sovelluksia käytetään enemmän kuin koskaan mobiililaitteilla.

Microsoftin ("Choose Between Traditional Web Apps and Single Page Apps (SPAs)" 2022), dokumentaatioissa mainitaan yhden sivun sovelluksen huonoiksi puoliksi hakukoneoptimointi, sivun navigointi ja suorituksen monitorointi

Hakukone etsii sisältöä esimerkiksi HTML-sivun uniikin otsikon, avainsanojen ja URL-osoitteen perusteella. Ongelmaa ei ole silloin, kun sovellusta käytetään esimerkiksi rajatussa ympäristössä,

kuten yrityksen sisäisessä verkossa. Hakukonebotti ei näe kaikkea sisältöä kerralla eikä voi näyttää sivua hakujen tuloksissa. Ongelman voi ratkaista tekemällä hakubotille oman sivun, tämä tosin voi olla haastavaa sivun monimutkaisuuden vuoksi.

Sovelluksen käynnistyessä huonosti optimoitu JavaScript sovelluskehysellään rakennettu sovellus voi olla todella kömpelö ja hidas.

Ongelma yhden sivun sovelluksessa voi olla sovelluksessa liikkuminen. Monisivuisessa sovelluksessa on helpompi liikkua, kun jokaisella sivulla on luonnostaan oma osoite.

2.2 Node.js & Express.js

Node.js on avoimenlähdekoodiin JavaScript suoritussympäristö, joka mahdollistaa JavaScript koodin suorittamisen palvelimella. Se on rakennettu Googlen kehittämän ChromiumV8-moottorin päälle.

Node.js on suosittu, koska sillä voidaan suorittaa JavaScript koodia palvelimella. JavaScriptin ollessa hallitseva koodikieli web-kehityksessä Node.js vakiinnutti paikkansa osana JavaScript tekniikoita.

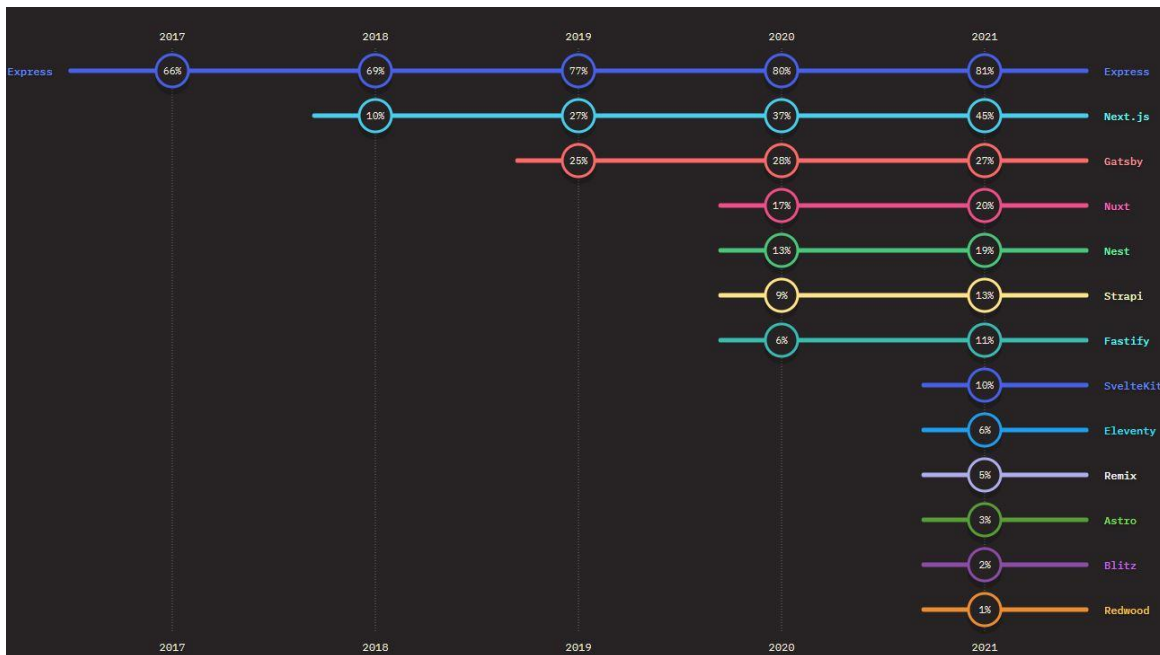
Nodeen on paljon sovelluskehyskiä. Tässä työssä käytetään yhtä suosituimmista Express.js. Tunnettuja sovelluskehyskiä ovat esimerkiksi Meteor.js, Sales.js, Koa.js, Keystone.js ja Loopback.js.

Noden ekosysteemiin kuuluu noden oma pakettienhallinta työkalu npm (node package manager). Pakettienhallinta työkalun avulla voidaan lisätä ja poistaa kirjastoja sovelluksesta. Kirjastot auttavat vähentämään koodin määrää. JavaScriptin käyttö palvelin- ja selainpuolella helpottaa kehittäjän työtä.

Noden sivuilla ("About Node.js", 2022) se määritellään tapahtumapainotteiseksi suoritussympäristöksi, joka suorittaa palvelimelle tulevia pyyntöjä koko ajan.

Tapahtumapainotteisuus tekee palvelimesta nopean, mutta jos pyyntö vaatii paljon laskentatehoa se voi aiheuttaa ongelmia. Liian suuren pyynnön tehdessä sovellukseen voi aiheutua viiveitä ja pyyntöjen kasautuessa sovellus ei toimi enää.

Suosituimmat palvelin puolen viitekehykset (Kuva 1, StateofJS, 2022-a).



Express.js on Noden sovelluskehys, joka helpottaa palvelinpuolen ohjelmointia. Express.js on tällä hetkellä suosituin sovelluskehys Nodelle State of JS-sivuston mukaan ("StateofJS" – 2022) Express on palvelinpuolen sovelluskehys. Ilman palvelinpuolen sovelluskehysten käyttöä joutuu tekemään paljon mutkikasta koodaamista ohjelmointirajapinnan eteen. Express ja muut sovelluskehykset ovat valmiiksi koodattuja kirjastoja, joista voidaan ottaa käyttöön haluttuja toimintoja vähäisellä koodin määrällä. Palvelinpuolen web-sovelluksen saadaan järjestettyä noudattamaan mallinäkömää-käsittelijä (Model-View-Controller) yksinkertaisemmin sovelluskehyksillä. Express on kevyt ja se on avoimen lähdekoodin projekti. Sen avulla on helppo hoitaa reitittäminen, sessiot, HTTP-pyyntöjä ja virhe käsittely.

2.3 Tietokanta

Sovelluksien tietojen tallentamiseen käytetään tietokantoja. Tietokanta voi olla paikallisen järjestelmänmuistissa tai erillisellä serverillä, johon sovellus ottaa yhteyden. Tietokannat voidaan jakaa SQL-relaatiotietokantoihin ja NoSQL-tietokantoihin. Perinteiset SQL-relaatiotietokannat noudattavat samoja standardeja toisin kuin NoSQL-tietokannat. NoSQL-tietokannoista on

olemassa erilaisia tyyppejä, kuinka tietoja varastoidaan ja käsitellään. Relaatietietokanta tarkoittaa relaatiomalliin perustuvaa tietokantaa.

Tietokantojen täytyy ratkaista ongelmat tietoturvaan, suorituskykyyn, pysyvyyteen ja eheyteen. Datan täytyy pysyä ehjänä esimerkiksi sähkökatkon yllättäessä. Kuinka varmistetaan, että dataa ei voi muokata ja poistaa saman aikaisesti.

Tietoturvalla tarkoitetaan rajattua saatavuutta. Miten tietokanta suojataan ja pääsy mahdollistetaan turvallisesti. Servereiden sijoittaminen maantieteellisesti on myös tärkeää. Tietokannan täytyy pystyä vastaamaan tehokkaasti kyselyihin ja tarjota haettava tieto mahdollisimman nopeasti.

Tietokannan täytyy pystyä suorittamaan tarvittavat kyselyt nopeasti. Onko sovellukselle välttämättä tärkeää, että sovellus näyttää kaikille samanlaiselta koko ajan vai vähitellen? Tietyissä sovelluksissa ei ole välttämätöntä, että sovellus päivittyy kaikille reaaliaikaisesti. Jos sovelluksen sisältö pitää olla kaikille reaaliaikaista, täytyy miettiä tarkoin mikä tietokanta valitaan ja miten tietokanta rakennetaan tehokkaaksi.

Tiedon pysyvyys. Esimerkiksi sähkökatkon sattuessa täytyy pystyä varmistamaan tiedon eheys. Tietoa ei saa päästä katoamaan ongelmatilanteen sattuessa.

Tietokannassa olevan datan täytyy säilyä ehjänä. Tietokantaan tallennettavan tiedon pitää täyttää asetetut määritelmät, että tieto tulee varmasti oikeaan muotoon. Esimerkiksi jos on määritelty, että syntymäaika sisältää pelkkiä numeroita täytyy varmistua, ettei muita merkkejä ole mahdollista päätyä tietokantaan.

Tietokantojen ylläpitoon on rakennettu tietokannanhallintajärjestelmiä. MySQL on esimerkiksi relaatiotietokantojenhallintajärjestelmä. Järjestelmällä on tarkoitus pitää yllä tietoturva, eheys, pysyvyys ja suorituskyky.

2.4 SQL-tietokanta

SQL (Structured Query Language) on 1970-luvulla kehitetty kyselykieli, joka mahdollistaa kyselyiden tekemisen tietokantoihin, määritellään Oraclen sivuilla ("What is database", 2022).

SQL-kielen avulla on mahdollista muokata, päivittää, lisätä ja poistaa tietoja tietokantatauluista. Moni tietokantahallintajärjestelmä tukee SQL-kieltä, mutta ohjelmien välillä on eroja.

Kaavio voi sisältää yhden tai useamman taulun. Taulu koostuu riveistä ja taulukossa on vähintään yksi sarake. Tauluissa on pääavain ja viiteavain. Näitä avaimia käyttämällä voidaan yhdistää tauluja toisiinsa.

Ero SQL ja NoSQL –tietokantojen välillä on niiden rakenne. SQL–tietokantaan lisättävä data on tarkasti ennalta määritelty, koska SQL-tietokannat noudattavat standardoituja malleja. SQL tietokannan vahvuuksia on monimutkaisten kysymyksien tekeminen useaan tauluun. Tunnisteella yhdistetään tauluja toisiinsa.

2.5 Tietokantahallintajärjestelmä

MySQL on maailman suosituimpia tietokantahallintajärjestelmiä. Oracle osti sen vuonna 2009, jonka jälkeen kaupalliseen käyttöön on tarvittu lisenssi. MySQL tukee useita tallennusmoottoreita esimerkiksi InnoDB ja MyISAM -tallennusmoottoreita. MySQL tukee paljon datatyyppejä ja pystyy käsittelemään useita yhteyksiä yhtä aikaa.

MariaDB on hyvin samankaltainen kuin MySQL. Oraclen ostettua MySQL muuttui maksulliseksi, osa kehittäjistä lähti kehittämään ilmaista ja parempaa versiota MySQL:stä. MariaDB on teknisesti paranneltu versio MySQL:stä. MariaDB:llä on enemmän tallennusmoottoreita, se on nopeampi, tukee suurempia yhteysaltaita ja pystyy tekemään näin ollen kaiken hieman nopeammin kuin MySQL. Se on ilmainen avoimeen lähdekoodiin perustuva tietokantojenhallintajärjestelmä.

SQLite on sulautettujärjestelmä. Esimerkiksi MySQL pyörii serverillä, johon client ottaa yhteyden, mutta SQLite toimii sovelluksessa itsessään. Näin ollen tietokantaan ei pääse käsiksi kuin sovellus itse paikallisesti. SQLite on kevyt eikä vaadi monimutkaisia konfiguraatioita. SQLite tukee vain rajallisen määrän datatyyppejä. SQLite sopii oikein hyvin yksinkertaisiin sovelluksiin tehokkaasti, todetaan SQLiten dokumentaatioissa ("About SQLite" - n. d.).

PostgreSQL on objekti-relaatiotietokanta. Se on suosittu yrityksissä, jotka haluavat määritellä tarkemmin tietokannan hakuja enemmän. PostgreSQL tukee molempia ei relaatio ja relaatio-datatyyppejä toisin kuin monet muut relaatiotietokanta hallintajärjestelmät. PostgreSQL tukee

myös muita kieliä kuin SQL. PostgreSQL kerrotaan IBM sivuilla ("What is PostgreSQL", 2019) sopivan monimutkaisempiin projekteihin, joissa täytyy tehdä paljon määrittäyksiä.

2.6 NoSQL-tietokanta

NoSQL termi on laajakäsite mikä tarkoittaa vaihtoehtoista tietokantaa perinteiselle SQL-tietokannalle. Relaatiotietokanta hallintajärjestelmät perustuvat kaikki samaan relaatiomallia noudattavaan standardiin. NoSQL-tietokantoihin puolestaan kuuluu lukuisia erilaisia teknisiä ratkaisuja datan säilömiseen.

NoSQL-ratkaisut kehittyivät 2000-luvun lopussa, kun datan määrä alkoi räjähdysmäisesti kasvaa sovelluksissa, kerrotaan MongoDB:n sivulla (Schaefer, L., n. d.). Yritykset kuten esimerkiksi Google ja Amazon alkoivat kehittää omia ratkaisuja suurien datamäärien käsittelyyn. Usein vaihtuva massiivinen data määrä ei ollut enää teknisesti järkevää yrittää toteuttaa relaatiotietokannoilla.

NoSQL-ratkaisuiden kehitystä vauhdittivat, myös itse sovelluskehityksen nopeutuminen NoSQL-tietokannoilla. Useiden erilaisten kaavioiden käsittely on helpompaa NoSQL-ratkaisuilla. NoSQL-tietokantoja voidaan skaalata usealle serverille, joka mahdollistaa suurien datan määrien käsittelyn väitetään MongoDB:n sivustolla (Schaefer, L., n. d.).

NoSQL-tietokanta ei seuraa mitään ennalta määritettyä rakennetta. Tietokantaan voidaan lisätä tietoa ilman, että tietokantaan tehdään ennakkoon perusteellisia määrittäyksiä. Tietokanta sopii useasti muuttuvaan tietokannan käsittelyyn. Tämä sopii hyvin yritykselle, jonka tietokanta on suuri ja tietorakenne muuttuu usein.

NoSQL-tietokantatyypit jaetaan karkeasti neljään kategoriaan. Avain-arvotietokanta, dokumenttitietokanta, saraketietotietokanta ja verkkotietokanta.

2.7 Document database

Dokumenttitietokanta (Document database) voi varastoida dataa JSON, BSON tai XML dokumentteina. Dataa voidaan varastoida lähes samassa muodossa kuin itse sovelluksessa, joka yksinkertaistaa sovelluskehitystä. SQL-dataa täytyy käsitellä erikseen siirrettäessä sovelluksen ja tietokannan välillä. Sovelluksen muuttuessa tietokantaa ei tarvitse määritellä uudelleen vaan

dataa voidaan säilöä edelleen, vaikka sovelluksen datan rakenne muuttuisi. Dokumenttitietokanta on erittäin suosittu kehittäjien keskuudessa, koska tietokannasta tulee ikään kuin osa koodia ja se on kehittäjän hallinnassa. SQL-tietokannassa muutoksia tietokantaan voidaan joutua käymään läpi erikseen tietokannanhallintajärjestelmän admin käyttäjän kautta, joka hidastaa kehitysprosessia merkittävästi. Dokumenttitietokanta on yleisesti käytetyin NoSQL-tietokantamuoto.

MongoDB on dokumenttipohjainen NoSQL-tietokanta ja se on erittäin suosittu web-sovelluskehityksessä. MongoDB:llä on mahdollista tallentaa JSON-muotoisia objekteja sovelluksesta suoraan muuttamatta datan muotoa erikseen.

Apachen CouchDB on dokumenttipohjainen avoimeen lähdekoodiin perustuva NoSQL-tietokanta. Datan varastointiin käytetään JSON-formaattia. JavaScript toimii kyselykielenä.

2.8 Key-Value database

Avain-arvotietokanta (Key-Value database) on hyvin yksinkertainen. Se sisältää vain avaimen ja arvon. Avain-arvotietokantoja käytetään yleensä toisena tietokantana, josta haetaan harvoin muuttuvaa yksinkertaista tietoa suuresta määrästä dataa. Käyttökohteet esimerkiksi ostoskori ja käyttäjän profiili.

Redis varastoi dataa avain arvo periaatteella. Esimerkiksi "Key:Name" ja "Value:Olli". Nämä muodostavat yhdessä avain-arvoparin. Redis käyttää RAM-muistia toisin kuin yleensä tietokanta käyttää levytilaa, joka tekee tietokannasta nopean, mutta samalla järjestelmän hidastuessa on vaara menettää tietoja. Redis-tietokantaa käytetäänkin pienien tietomäärien tallentamiseen. Se ei sovellu suuriin kokonaisuuksiin sen arvo avain periaatteen vuoksi. Sitä käytetään yleensä toisena tietokantana sisältämään dataa, joka muuttuu harvoin. Redis-tietokanta ei tue XML-dataformaattia toisin kuin esimerkiksi MySQL.

2.9 Column-oriented database

Saraketietokanta (Column-oriented database) tallentaa tiedot sarakkeisiin. SQL-tietokanta tallentaa tiedot tauluun riville, joita käydään läpi rivi kerrallaan. Sarake pohjaisessa tietokannassa käydään vastaavasti läpi sarakkeita. Käytetään esimerkiksi datan analysoimisessa. Soveltuu tietyn mallisen datan käsittelyyn paremmin sarakkeiden ansiosta.

Apache Cassandra on suosittu saraketietokanta, joka perustuu avoimeen lähdekoodiin. Suuret yritykset kuten Netflix käyttävät Cassandra-tietokantaa kriittisten tietojen säilyttämiseen. Cassandra on nopea ja luotettava. Cassandra-tietokannassa ei ole master nodea, joten se on helppo skaalata ja varmuuskopiointi on turvallista.

2.10 Graph database

Verkkotietokanta (Graph database) tallentaa datan nodeina ja nodeja yhdistää edge. Yhdellä nodella voi olla useita yhteyksiä. Data mitä verkkotietokantaan tallennetaan voi olla esimerkiksi sosiaalisen medianprofiili. Verkkotietokanta keskittyy data elementtien välisiin suhteisiin. Käytetään esimerkiksi tunnistaman huijauksia ja linkittämään ihmisiä sosiaalisissa verkostoissa. Verkkotietokanta on yleensä toinen tietokanta jonkun perinteisemmän tietokanta ratkaisun yhteydessä. Verkkotietokanta on ideaalinen käsitellessä dataa, joka on itsenäinen, mutta riippuvainen useaan suuntaan.

Neo4j on yksi suosituimpia verkkotietokantahallintajärjestelmiä. Se esittää datan nodeina, joilla luodaan yhteyksiä tietokannan sisällä. Neo4j tukee useita koodi kieliä.

2.11 Sovelluskehykset

Web-sovelluskehityksessä käytetään JavaScript sovelluskehyskiä. Sovelluskehys on kokoelma JavaScript koodikirjastoja. Sovelluskehys sisältää valmiiksi koodattuja toimintoja, jotka nopeuttavat ja helpottavat monimutkaisten sovelluksien kehittämistä. Suosituimmat JavaScript -sovelluskehukset ovat React.js, Angular.js ja Vue.js ("StateofJS", 2022). Kirjastoja käytetään modernissa web-sovelluskehityksessä.

On mahdollista tehdä sovelluksia ilman edellä mainittuja sovelluskehyskiä, mutta silloin koodia täytyy kirjoittaa paljon enemmän. Loppujen lopuksi ajaudutaan tekemään oma sovelluskehys ja sitä kukaan ei halua. On hyvä tietää, milloin oikeasti tarvitsee sovelluskehystä ja milloin olisi hyvä käyttää puhdasta JavaScriptiä asioiden hoitamiseen perinteisellä web-sivulla.

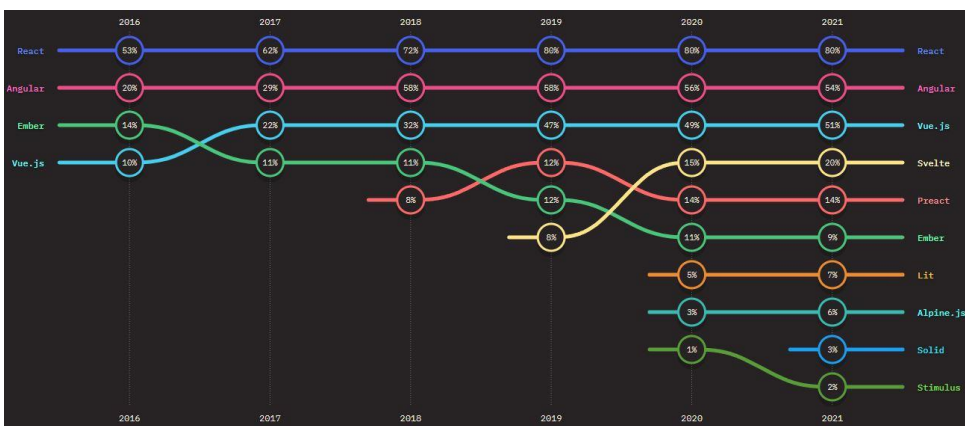
Sovelluskehys vai kirjasto? Välillä eroa vaikea määritellä. Ennen sovelluskehyskiin perehtymistä on muutama asia mikä kannattaa ottaa haltuun. Perinteiset HTML, CSS ja JavaScript. Osaaminen DOM

(Document Object Model). JavaScriptin versio ES6 (ECMAScript 2015, ES2015) ja Node.js ekosysteemi eli npm (Node Package Manager).

HTML (HyperText Markup Language) on kieli, jolla määritellään web-sivun rakenne. CSS (Cascading Style Sheets) kielen avulla määritellään HTML-dokumentin ulkoasua. DOM (Document Object Model) on HTML-tiedostossa toimiva rajapinta, joka määrittää rakenteen ja sisällön websivulla. Se kommunikoi koodikielen kanssa ja määrittää kuinka sivu reagoi.

JavaScript on 1995 julkaistu koodikieli, joka on yksi suosituimpia koodikieliä web-sovelluskehityksessä. JavaScriptiä voidaan suorittaa palvelimella, joka on räjäyttänyt suosion entisestään. JavaScript kehittyi merkittävästi ES6 versiossa vuonna 2015. JavaScriptin perusteet on hyvä osata ennen sovelluskehitysiin perehtymistä.

Suosituimmat palvelin selainpuolen viitekehykset (kuva 2, StateofJS, 2022).



Angular on Googlen kehittämä ensimmäinen sovelluskehys. Angular on raskain, koska se sisältää paljon valmiiksi koodattuja toimintoja. Siksi se muistuttaa enemmänkin sovellusalustaa kuin sovelluskehystä.

Vue.js on vapaaehtoisten ja Patreon-rahoituslustoja kautta rahoitettu avoimenlähdekoodin sovelluskehys. Vue on aloittelijalle hyvä valinta, koska se on helppo ottaa käyttöön verrattuna esimerkiksi Angulariin. Sen lähestymistapa on perinteisempi html, js ja css omina tiedostoinaan toisin kuin React-sovelluksissa jsx ja javascript ovat keskeisemmässä osassa todetaan tutorialspoint-sivuston dokumentaatiossa ("Vue.js – Overview", 2022)

2.12 React.Js

React.js on vuonna 2013 Facebookin kehittämä JavaScript-kirjasto, joka helpottaa käyttöliittymän rakentamista web-sovelluksille. Reactia pidetään enemmän kirjastona, kuin sovelluskehiksenä verrattuna Angulariin. Se on vakiinnuttanut paikkansa suosituimpana sovelluskehiksenä stateofjs("StateofJS", 2022) ja npm trends ("npm trends", 2022) sivustoiden mukaan.

React kiteytyy sovelluksen jakamiseen komponentteihin, niiden tilanhallinta ja tiedon välittäminen komponenttien välillä. Sovellus pilkotaan erillisiin tiedostoihin mukailemaan MVC-sovellusarkkitehtuuria. React-sovellus on kuin kokoelma kirjastoja, joista sovellus koostuu. React-sovelluksissa koodia kirjoitetaan pääosin JavaScriptiä ja JSX-tekniikkaa käyttäen. HTML ja CSS eivät ole niin keskeisessä roolissa kuin esimerkiksi Vue.js-sovelluskehiksessä.

Angular on hieman raskaampi ja siihen on enemmän sisään rakennettuja toimintoja verrattuna Reactiin. Vue.js asettuu tähän väliin olemalla hieman raskaampi kuin React, mutta ei niin kehitysalustamainen kuin Angular.

Alkuun Reactilla pääsee nopeasti luomalla sovelluksen kirjoittamalla komentoriville `npx create-react-app myApp`. Viimeinen `myApp`-komento on sovelluksen nimi, joka muodostaa `myApp` nimiseen kansioon sovelluksen rungon valmiiksi. UI/DOM rajapinnan ja tilan hallinta on kaikissa kolmessa sovelluskehiksessä rakennettu sisään. Eroavaisuuksia tulee reitityksessä, lomakkeiden käsittelyssä ja http-pyyntöjen tekemisessä. Tiivistettynä Angular on raskain, React on kevyin ja Vue asettuu näiden kahden välille.

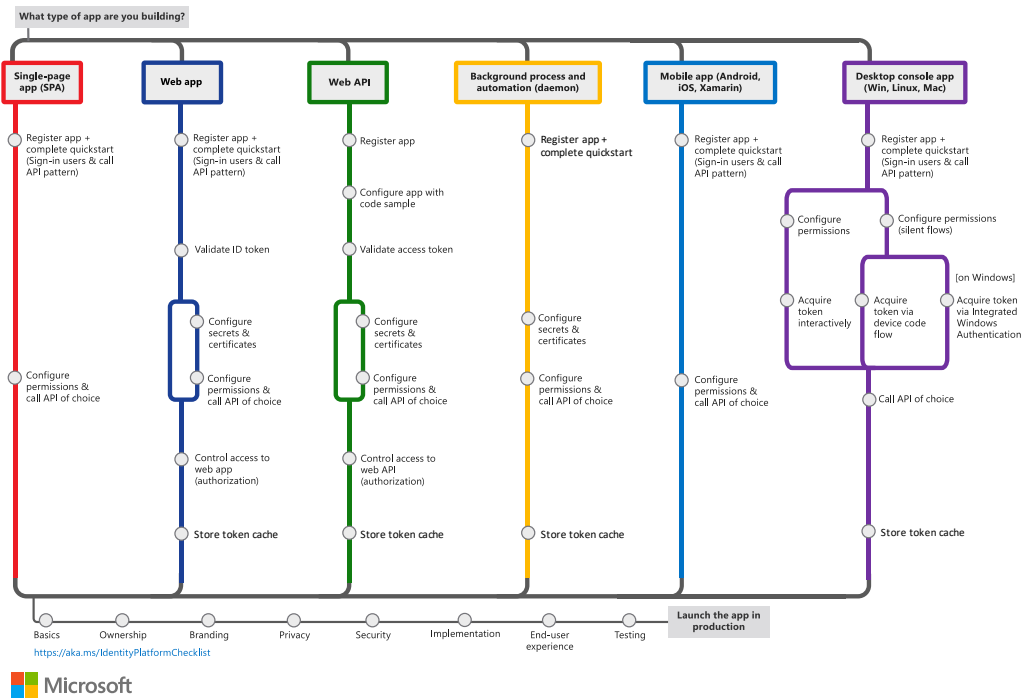
2.13 Microsoft Identity Platform

Microsoft Identity Platform on Microsoftin kehittäjille rakennettu alusta, joka helpottaa yhdistämään sovelluksiin luotettavan tunnistautumistavan. Sen avulla sovelluksista saadaan turvallisia ja vältetään useiden monimutkaisten käyttäjienhallinta ratkaisuiden luomista. Identity Platform ja Graph -palveluiden avulla saadaan avattua rajapinta käyttäjän henkilökohtaisiin tietoihin. Käyttäjän tietoihin päästään käsiksi http-pyyntöjä tekemällä rest-api-rajapintaan. Microsoft Graph on ohjelmointi rajapinta, josta saadaan http-pyyntöjä tekemällä käyttäjän tietoja Microsoftin pilvipalvelusta Office 365:sta.

Microsoft identity platform toiminta kaaviossa (kuva 3, "Microsoft identity platform", 2022).

Microsoft identity platform

<http://aka.ms/IdentityPlatform>



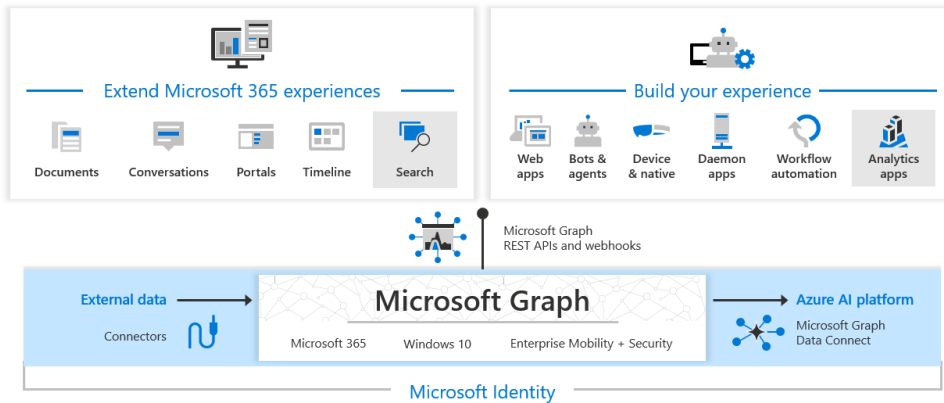
Microsoft Identity Platform koostuu muutamasta eri komponentista. OAuth 2.0 ja OpenID Connect mahdollistavat kehittäjien useiden eri sosiaalisten tilien yhdistämisen sovellukseen. Microsoft Authentication Libraries (MSAL) eli avoimen lähdekoodin tunnistautumiskirjastot. Microsoft Azure portaalissa sovelluksen hallinta. Sovellus rekisteröidään ja määritellään asetukset. Sovelluksen API-rajapinnan määrittely ja sovelluskehitystä voidaan automatisoida Power Shell työkalulla.

Microsoftin identity platform tuo mukanaan sovelluksen käyttöön salasanonnattoman tunnistautumisen, vaiheittaisen tunnistautumisen ja ehdollisen pääsyn, joten niiden integroiminen sovellukseen on helppoa.

2.14 Microsoft Graph

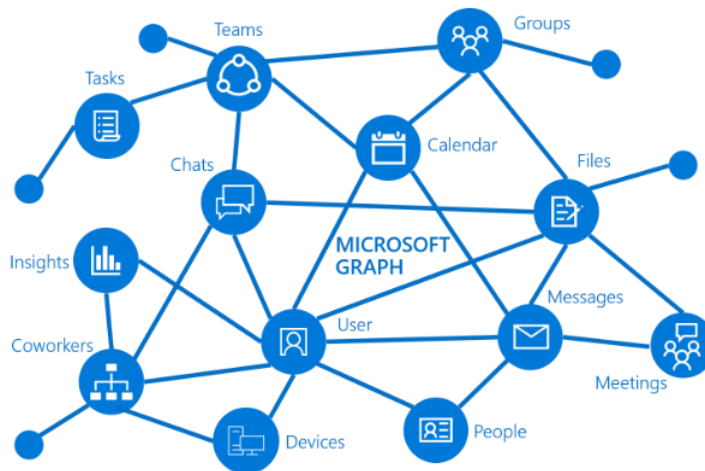
Microsoft Graph palvelun rooli sovelluksessa (kuva 4, ”Microsoft Graph”, 2022).

Microsoft 365 Platform



Microsoft Graph on ohjelmointi rajapinta käyttäjän Office 365 dataan. Graph –palvelun avulla voidaan luoda älykkäitä sovelluksia, jotka kommunikoivat Microsoftin pilvipalvelun kanssa.

Microsoft Graph palveluiden yhdistäminen (kuva 5, ”Microsoft Graph”, 2022).



Dataa voidaan hakea, lisätä, päivittää ja poistaa Graph-palvelun avulla. Graph-palvelu sisältää työkalut käyttäjien ja laitteiden hallintaan, käyttöoikeuksiin, turvallisuuteen ja datan säilyvyyteen. Rajapinnan avulla päästään käsiksi esimerkiksi käyttäjän Outlook-kalenteri merkintöihin, profiilikuvaan ja OneDrive tiedostoihin. Graph-rajapinnan opettelu suoraviivaistaa kehitystyötä, koska enää ei tarvitse opetella erikseen Outlook-rajapinnan käyttöä kerrotan Microsoftin

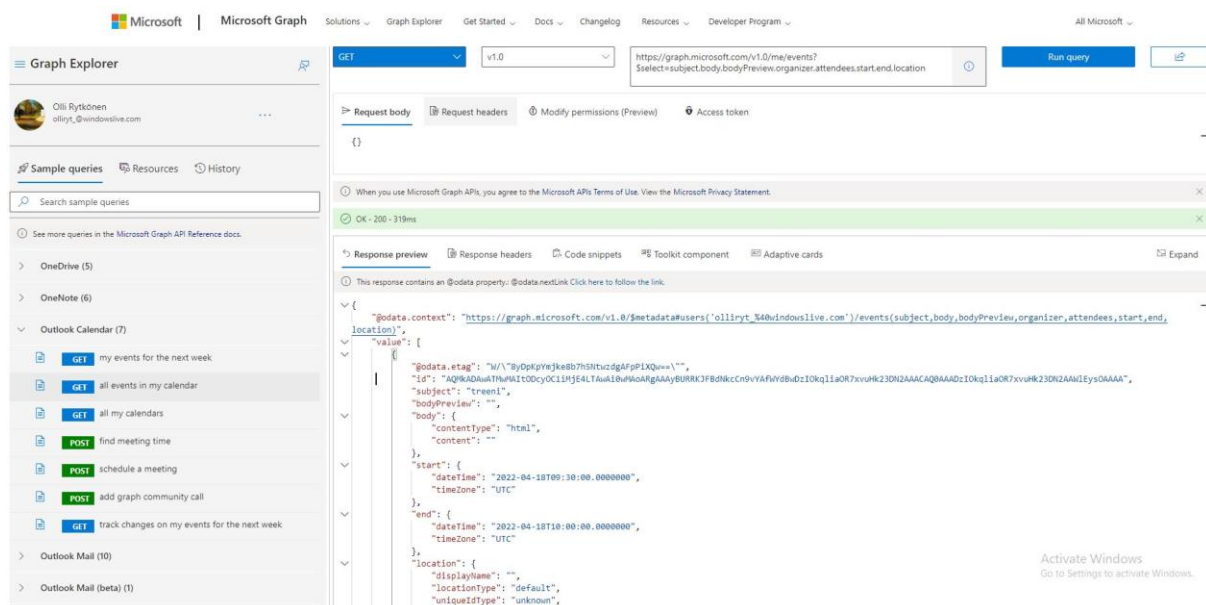
dokumentaatioissa ("Roles in OAUTH 2.0, Tokens, App registration, Endpoints", 2022). Käyttäjän tietoja haetaan yksinkertaisesti http-pyyntöjä tekemällä.

Tunnistautuminen Graph-palvelua käyttäen SPA-sovelluksessa (kuva 6, "How the sample works", 2022).



OAuth 2.0 (Open Authorization) on tunnistautumis- protokolla, joka on suunniteltu web-sovelluksien keskinäiseen tunnistautumiseen. Käytetään esimerkiksi rajapintoihin kiinni pääsemiseen. Sovellus ei vaadi ulkoisia tunnistautumismenetelmiä, vaan kaikki tapahtuu sisäisesti tokenien avulla. OAuth 2.0 on uudelleen kirjoitettu OAuth 1.0 versiosta, eikä ole yhteensopiva vanhempaan versioon.

Microsoft Graph Explorer kehittäjille suunnattu työkalu (kuva 7, "Microsoft explorer", 2022).



Microsoftin dokumentaatioissa ("Roles in OAUTH 2.0, Tokens, App registration, Endpoints", 2022) määritellään OAuth 2.0 koostuvan tunnistautuminen koostuu resurssin omistajasta, clientistä, resurssi serveristä ja Microsoft identity platformista eli tunnistautumis- serveristä. Tunnistautumis-serverillä käsitellään turvallisesti käyttäjän tunnistautuminen token-tunnisteilla, jotka on luotu web-sovelluksessa ja API-rajapinnassa. Client eli web-sovellus, jota käyttäjä käyttää. Resurssin omistajalla tarkoitetaan loppukäyttäjää, joka omistaa oman datan, jolla tunnistautuminen suoritetaan. Käyttäjä pystyy aina päättämään antaako lupaa sovellukselle käyttää omaa dataa. Resurssi-serverillä tarkoitetaan serveriä, jolla säilötään käyttäjän omistamaa dataa.

Tokenoja käytetään turvalliseen tunnistautumiseen vakuudeksi, että käyttäjä, resurssi tai palvelu on varmasti oikea. Identity platform -tokenit ovat JSON-muodossa (JSON Web Tokens). Tunnistautumisalusta käyttää kolmea token-tyyppiä: Acces tokens, ID tokens ja refresh tokens.

Acces tokens luodaan tunnistautumisserverillä. Client siirtää Acces tokenin resurssiserverille. ID tokens sisältää informaation käyttäjästä ja se muodostetaan clientin-sovelluksessa. Refresh tokens välittää tietoa clientilta tunnistautumisserverille ("Roles in OAUTH 2.0, Tokens, App registration, Endpoints", 2022).

Sovellus täytyy rekisteröidä Azuren-pilvipalveluun, jolla saadaan sovellukselle omatunniste (Application ID, client ID). Client ID:tä käytetään koodissa, kun luodaan yhteyttä tunnistautumiseen. Toinen vaihtoehto on käyttää Redirect URI, joka tunnistautumisen jälkeen ohjaa käyttäjän toiseen osoitteeseen. Sovelluksen asetuksissa Azuren-pilvipalvelussa määritellään mitä tietoja sovellus pyytää käytettäväksi ja minne käyttäjä ohjataan tunnistautumisen jälkeen ("Roles in OAUTH 2.0, Tokens, App registration, Endpoints", 2022). Omaan sovellukseen oma url-osoite generoituu sovelluksen rekisteröimisen jälkeen, jota käytetään tunnistautumisen jälkeisessä sivulle ohjaamisessa. Microsoft Graph Explorer on kätevä työkalu alkuun pääsemiseksi kehittämisessä.

2.15 Koodieditorit ja kehitysympäristöt

Koodieditorin ja kehitysympäristön keskeisin ero on niiden ominaisuudet. Koodia muokkaamaan riittää tekstinkäsittelyohjelma, joten koodieditorit ovat suhteellisen kevyitä ohjelmia.

Kehitysympäristö on puolestaan tarkoitettu monimutkaisiin sovelluksien suorituksiin, joten ne ovat lähtökohtaisesti todella raskaita niiden lukuisten toimintojen vuoksi.

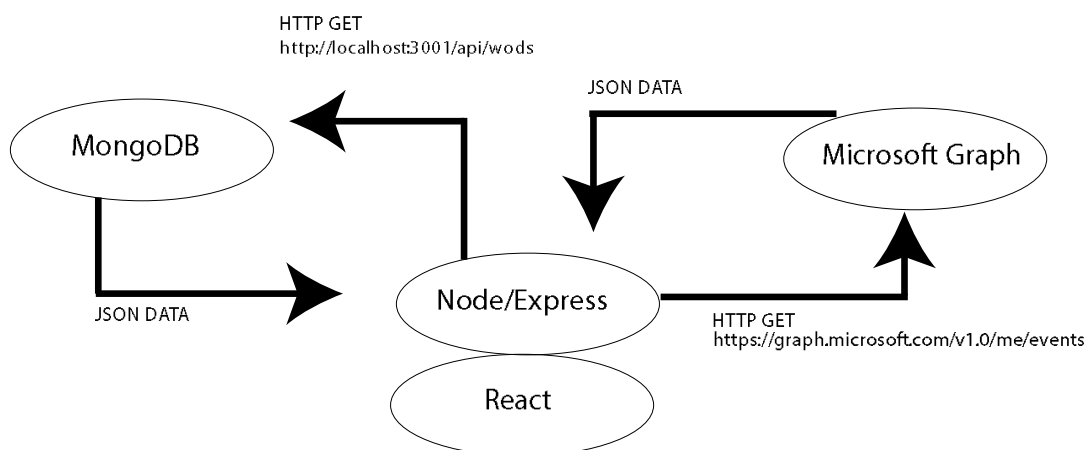
Visual Studio Code on suosituin koodieditori Top IDE Index ("Top IDE index", 2022) sivuston mukaan. Se on ilmainen ja kevyt selaimen päälle rakennettu koodieditori. Visual Studio Code:ssa on sisään rakennettu Git-versionhallintajärjestelmän tuki. Editoriin voi lisätä kirjastoja helpottamaan koodin kirjoittamista ja testaamista. Sovellus keskittyy koodin testaaminen ja ylläpitämiseen, mutta monimutkaiset toiminta vaiheet on jätetty Visual Studioon, joka on sovelluskehitysympäristö. Visual Studiossa Codella voidaan ketterästi testata koodia. Editoriin voidaan ladata aina kielikohtaisia testaustyökaluja ja kirjastoja. Esimerkiksi ennustavaa koodin kirjoittamista.

Sovelluskehitys ympäristöt ovat luonnollisesti raskaampia kuin koodieditorit. Suosituimpia ovat esimerkiksi Visual Studio, Eclipse ja IntelliJ. Suosio vaihtelee koodikielen mukaan. Android studio on luonnollisesti Android-kehitykseen. Eclipse ja NetBeans ovat suosittuja Java-kehittäjien keskuudessa. Python kehitykseen on omat suositut ympäristöt Vim, Atom , Sublime ja PyCharm.

3 Toteutus

Tarkoituksena on luoda yksinkertainen treenisovellus, josta voidaan lisätä käyttäjän henkilökohtaiseen Outlook-kalenteriin merkintä. Koodissa käytetään paljon crossfitlajista tuttua termiä WOD, joka on lyhenne päivän treenistä (Work Out of the Day).

Sovelluksen data liikenne MongoDB-tietokantaan ja Microsoftin pilvipalveluun (kuva 8).



Palvelinpuoli tehdään node-ympäristöön ja selainpuoli tehdään React-kirjastoa hyödyntäen.

Lopuksi otetaan käyttöön Graph-palvelu.

Koodaamiseen opinnäytetyössä käytetään Visual Studio Code -koodieditoria, koska se on tuttu entuudestaan ja erittäin suosittu.

3.1 Nodejs/Express

Palvelinpuoli toteutetaan Node-suoritusympäristössä Express-sovelluskehystä käyttäen. Visual Studio Code sisältää komentorivin, jolla voidaan suorittaa tarvittavat komennot npm-komentojen avulla.

Komennolla "npm init" luodaan projektin asetukset määrittelevä tiedosto ja tehdään tarvittavat määrittelyt.

```
{
  "name": "cfapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "server": "nodemon index.js",
    "start": "node index.js"
  },
  "author": "Olli Rytönen",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^14.2.0",
    "express": "^4.17.2",
    "mongoose": "^6.1.7"
  },
  "devDependencies": {
    "nodemon": "^2.0.15"
  }
}
```

Tiedostossa määritellään sovelluksen asetukset. Keskeisimmät skriptit ovat sovelluksen käynnistämiseen, testaamiseen ja riippuvuuksiin liittyvät määrittelyt. Sovelluksen runko rakentuu index.js tiedoston.

Riippuvuudet eli sovellukseen lisättävät JavaScript-koodikirjastot asennetaan npm komennoilla komentoriviä käyttäen projektin juuressa. Esimerkiksi npm install cors. Projektiin voidaan määritellä kehityksen aikaisia devDependencies-kirjastoja kuten palvelinpuolen testaamista helpottamaan asennettu nodemon-kirjasto, joka automaattisesti käynnistää palvelimen uudelleen, kun muutokset tallennetaan.

Node-sovelluksen toiminnot eritellään omiin kansioihin. Sovelluksen toiminnallisuus pidetään tässä projektissa kuitenkin pelkästään index.js tiedostossa yksinkertaisuuden vuoksi.

Käydään läpi sovelluksen kansiorakenne. Build kansioon tuodaan React-sovelluksesta rakennettu build-kansio. Node palvelinpuoli ottaa käyttöön sovelluksen selainpuolen build kansiota. Models kansioon määritellään tietokantaan tehtävä yhteyden määrittely.

"Node_modules" -kansioon asentuu node-kirjastot. Tiedostossa "gitignore" määritellään mitä tiedostoja ei haluta lisätä git-versionhallintaan. Esimerkiksi ".env"-tiedosto, jossa määritellään sovelluksen kriittisiä asetuksia. "Node_modules"-kansio asennetaan, kun projekti ladataan githubista ja asennetaan npm install -komennolla. Silloin asentuu package.json tiedostossa määritellyt riippuvuudet.

3.2 Node/Express sovelluksen runko

Sovelluksen toiminnallisuus on rakennettu index.js tiedostoon. Käydään läpi index.js tiedoston sisältö.

```
const express = require('express')
const app = express()
const cors = require('cors')
require('dotenv').config()
const Wod = require('./models/wod')
app.use(cors())
app.use(express.json())
app.use(express.static('build'))
```

Ensin määritellään sovellukselle keskeiset muuttujat. Otetaan käyttöön Express luomalla app-muuttuja. App-muuttujan avulla otetaan käyttöön cors ja express.json komennoilla app.use. App-muuttujan luomisen jälkeen otetaan käyttöön cors-kirjasto ja luodaan sille oma muuttuja. Otetaan käyttöön "dotenv" -tiedosto. Tiedosto sisältää sovelluksen määrittelyihin liittyviä osia, joita ei haluta julkaista koodin mukana.

Sovelluksen runkoon määritellään rajapintaan kohdistuvien pyyntöjen käsittely.

Ohjelmointirajapinta seuraa polkua /api/wods. Wods muuttuja sisältää kaikki treenit ja Wod tarkoittaa yksittäistä treeniä.

3.3 REST-rajapinnan luominen

```
app.get('/api/wods', (request, response) => {
  Wod.find({}).then(wods => {
    response.json(wods)
  })
})
```

Get komento hakee kaikki kyseisestä rajapinnasta olevan datan. Wod.find komento hakee koko listan, jonka jälkeen palauttaa sen JavaScript-objektina wods.

```
app.post('/api/wods', (request, response) => {
  const body = request.body
  const wod = new Wod({
    subject: body.subject,
    start: {
      dateTime: body.start.dateTime,
      timeZone: body.start.timeZone,
    },
    end: {
      dateTime: body.end.dateTime,
      timeZone: body.end.timeZone,
    }
  })
  wod.save().then(savedWod => {
    response.json(savedWod)
  })
})
```


Post-pyyntö lisää uuden treenin tietokantaan. Luodaan muuttuja body, joka ottaa datan vastaan JSON-muodossa. Muuttujaan wod määritellään index.js tiedostoon tuotu models kansiota Wod kaavion määrittely tiedon lisäämiseksi MongoDB-tietokantaan. Wod tarkoittaa uutta treeniä. Tietokantaan yhdistämiseen määrittelyt ovat omassa tiedostossa. Sen jälkeen määritellään treenin nimi ja ajankohta.

```
subject: body.subject,
  start: {
    dateTime: body.start.dateTime,
    timeZone: body.start.timeZone,
  },
  end: {
    dateTime: body.end.dateTime,
    timeZone: body.end.timeZone,
  }
}
```

Myöhemmin lisättävään Office365-palveluun määrittelyt vaativat datan olevan kyseisessä muodossa. Subject kuvaa wodin eli treenin kalenterimerkinnän nimen. Ajankohdan lisäämiseen Office365-palveluun tarvitaan aika ja aikavyöhyke. Aikavyöhyke on kovakoodattu, koska ongelmia ilmeni ajan määrittämisen kanssa, joten ajan säästämiseksi päätin kova koodata aikavyöhykkeen. Kovakoodaaminen tarkoittaa, että käyttäjä ei voi määritellä aikavyöhykettä vaan se on ennalta päätetty.

```
wod.save().then(savedWod => {
  response.json(savedWod)
})
```

Lopuksi wod.save toiminnolla tallennetaan savedWod-objekti ja lähetetään palvelimelle.

```
app.put('/api/wods/:id', (request, response) => {
  const body = request.body

  const wod = {
    subject: body.subject,
    start: {
      dateTime:body.dateTime,
      timeZone:body.timeZone
    },
    end:{
      dateTime:body.dateTime,
      timeZone:body.timeZone
    }
  }
})
```

```

Wod.findByIdAndUpdate(request.params.id, wod, { new: true })
  .then(updatedWod => {
    response.json(updatedWod)
  })
})

app.delete('/api/wods/:id', (request, response) => {
  Wod.findByIdAndRemove(request.params.id)
  .then(result => {
    response.status(204).end()
  })
})

```

Tiedon päivittäminen tietokantaan put-metodia käyttäen tapahtuu hyvin samankaltaisesti kuin tiedon lisääminen post-metodilla. Funktion sisäiseen muuttujaan wod lisätään tarvittavat määrittelyt, jonka jälkeen kuvioon astuu Express-kirjaston toiminnallisuus findByIdAndUpdate.

Tarkastellaan findByIdAndUpdate-funktiota tarkemmin. Funktio saa kaksi parametria request.params.id ja wod. Sovellus etsii vastaavan id-numeron tietokannasta ja korvaa vanhan objektin uudella.

Tiedon poistamiseen käytettävä delete-funktio on hyvin yksinkertainen. Funktio tarvitsee id:n jolla poistettava kohde etsitään tietokannasta. Delete-pyyntö tehdään osoitteeseen /api/wods/:id rajapintaan. Express tarjoaa kohteen poistamiseen valmiin toiminnallisuuden findByIdAndRemove. Onnistunut poistaminen palauttaa vastauksen "status(204)" serveriltä.

```

const PORT = process.env.PORT || 3001
app.listen(PORT, () => {
  console.log(`Listening port: ${PORT}`)
})

```

Tiedoston index.js viimeinen osio on sovelluksen portin määrittely. Sovellus kuuntelee porttia, joka on määritelty ".env"-tiedostossa, jos sitä ei löydy niin kuunnellaan porttia numero 3001.

Sovelluksen API-rajapinta (Kuva 9).

Selain

Palvelin

| | | |
|-------------|------------------------------------|---|
| HTTP GET | http://localhost:3001/api/wods | Palauttaa kaikki objektit wods json muodossa rajapinnasta |
| HTTP POST | http://localhost:3001/api/wods | Lähetää uuden objektin wods rajapintaan |
| HTTP PUT | http://localhost:3001/api/wods/ id | Korvaa edellisen objektin uudella |
| HTTP DELETE | http://localhost:3001/api/wods/ id | Palauttaa statuksen 204, kun poisto onnistuu |

3.4 MongoDB määrittely

MongoDB Atlas pilvitietokantaan sovellukselle tietokanta luodaan selaimella. Määrittelyissä valitaan ilmainen tilaus ja konesali mahdollisimman läheltä maantieteellisesti. Sen jälkeen määritellään käyttäjätunnus, jolla otetaan yhteystietokantaan. Tietokantaan pääsemiseen on monia tapoja, mutta tässä projektissa käytetään IP-osoitetta sen yksinkertaisuuden vuoksi.

Yhteyden luomiseen käytetään Mongoose-kirjastoa, jonka avulla JavaScript-olioiden tallentaminen on suoraviivaista. Tietokanta määritellään sovelluksessa omaan tiedostoon.

```
const mongoose = require('mongoose')
const url = process.env.MONGODB_URI

console.log('Connecting to MongoDB', url)
mongoose.connect(url, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(result => {
    console.log('connected to MongoDB')
  })
  .catch((error) => {
    console.log('error connecting to MongoDB:', error.message)
  })

const wodSchema = new mongoose.Schema({
  subject: String,
  start: Array,
  end: Array
})
```

```
wodSchema.set('toJSON', {
  transform: (document, returnedObject) => {
    returnedObject.id = returnedObject._id.toString()
    delete returnedObject._id
    delete returnedObject.__v
  }
})

module.exports = mongoose.model('Wod', wodSchema)
```

Tiedoston alkuun määritellään Mongoose käyttöön ja MongoDB Atlas palvelusta haettu URL-osoite. Oleellista on kaavion määrittely wodSchema, jossa määritellään tietokantaan lähetettävä datan muoto.

Nyt olemme luoneet Node-sovelluksen pohjan, joka on yhteydessä tietokantaan ja tarjoaa REST-rajapinnan selain puolen käyttöön.

3.5 React sovelluksen toteuttaminen

React-sovellus aloitetaan tekemällä oma kansio erilleen Node kansista ikään kuin oma projekti. React-sovelluksen voi luoda ilman valmista npx create-react-app komentoa. Komento luo paljon valmiita ja osittain turhia tiedostoja, jotka siivotaan pois. React-sovelluksessa on sovelluksen runkotiedosto App.js, joka kokoaa tarvittavat toiminnot yhteen. Koodia jaetaan omiin komponentteihin toimintojen perusteella.

React-sovelluksessakin sovelluksen määrittelyt löytyvät package.json tiedostossa, jossa päästään käsiksi sovelluksen riippuvuuksiin.

```
"dependencies": {
  "@azure/msal-browser": "^2.21.0",
  "@azure/msal-react": "^1.2.0",
  "@microsoft/mgt-element": "^2.3.1",
  "@microsoft/mgt-msal2-provider": "^2.3.1",
  "@microsoft/mgt-react": "^2.3.1",
  "@testing-library/jest-dom": "^5.16.1",
  "@testing-library/react": "^12.1.2",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^0.25.0",
  "bootstrap": "^5.1.3",
  "dotenv": "^14.2.0",
  "react": "^17.0.2",
```

```

    "react-bootstrap": "^2.1.1",
    "react-dom": "^17.0.2",
    "react-router-bootstrap": "^0.25.0",
    "react-router-dom": "^5.3.0",
    "react-scripts": "5.0.0",
  }

```

Kaikki riippuvuudet asennetaan npm install -komennolla. React-sovelluskehys otetaan käyttöön kirjastona niin kuin muutkin JavaScript kirjastot.

Sovellukseen asennetaan Microsoft Azureen liittyvät valmiit kirjastot, joilla tunnistautuminen ja sovelluksen liittäminen Office365-pilvipalveluun tapahtuu. Axios-kirjasto helpottaa reitittämistä pyyntöjen tekemiseen node-sovelluksessa luomaamme rajapintaan. Bootstrap ja React-bootstrap kirjastot auttavat valmiiden komponenttien käyttöön ottamista sovelluksen ulkoisen ilmeen parantamiseksi.

Kansio rakenteessa src-kansio muodostaa sovelluksen resurssit, minne tulevat komponentit, kuvat ja sovelluksen App.js runkotiedosto.

App.js tiedostoon määritellään kirjastot ja tuodaan itse kasatut komponentit.

3.6 Sovelluksen määrittelyt

```

import React, { useState, useEffect } from 'react'
import 'bootstrap/dist/css/bootstrap.min.css';
import Container from 'react-bootstrap/Container';
import Navbar from 'react-bootstrap/Navbar';
import Nav from 'react-bootstrap/Nav';

import {Providers} from '@microsoft/mgt-element';
import {Msal2Provider} from '@microsoft/mgt-msal2-provider';
import {Login} from '@microsoft/mgt-react';

import Home from '../src/components/Home'
import Wods from '../src/components/Wods'
import Info from '../src/components/Info'
import UserCalendar from '../src/components/UserCalendar'

import wodService from '../services/wods'
import {
  BrowserRouter as Router,
  Switch,
  Route,

```

```

    Link
  } from "react-router-dom";

  Providers.globalProvider = new Msal2Provider({
    clientId: "tähän oma clientID",
    scopes: ['calendars.readwrite', 'user.read', 'openid', 'profile',
'people.read', 'user.readbasic.all']
  });

```

Tuodaan tarvittavat kirjastot ja komponentit. Sivun eri osat on jaettu omiin komponentteihin kuten navigointi palkki ja kalenterit.

3.7 Sovelluksen runko

```

const App = () => {

  const [wods, setWods] = useState([])
  const [wodTitle, setWodTitle] = useState('')
  const [newStartTime, setStartTime] = useState('')
  const [newEndTime, setEndTime] = useState('')
  const timeZone = 'UTC'

```

Sovellus itsessään on nuolifunktio, joka tiedoston lopussa tuodaan eteenpäin App funktiona. Muodostetaan siis const App -nuolifunktio ja funktioon määritellään treenejä koskevat muuttujat. Kaikki treenit omana taulukkona wods. Uuden treenin lisäämiselle määritellään nimi, aloitus- ja lopetus aika. UseState on react-kirjastosta tuotu toiminto, jolla voidaan hallita muuttujan tilaa. UseState on aina sen hetkinen tila, kun uutta treeniä ollaan luomassa.

```

  useEffect(() => {
    wodService
      .getAll()
      .then(response => {
        setWods(response.data)
      })
  }, [])

```

UseEffect-ominaisuutta käytetään tuomaan kaikki treenit tietokannasta. Tiedostosta wodService tuodaan getAll funktio, joka asettaa setWods toiminnolla wods taulukkoon json-muotoisen datan.

```

const addWod = (event) => {
  event.preventDefault()
  const wodObject = {
    subject: wodTitle,
    start: {

```

```

        dateTime: newStartTime,
        timeZone: timeZone
      },
      end: {
        dateTime: newEndTime,
        timeZone: timeZone
      },
    }
    setWodTitle('')
    setStartTime('')
    setEndTime('')
    Providers.globalProvider.graph.client.api('/me/events')
      .post(wodObject)
  }

```

Treeniin lisääminen tietokantaan ja M365 kalenteriin tapahtuu funktiossa addWod.

Event.preventDefault() funktio estää sivun uudelleen päivittymisen. Luodaan funktion sisäinen muuttuja wodObject, joka ottaa vastaan halutut arvot. Lopuksi muuttujien tilat muutetaan oletusarvoisiksi ja Providers globaali muuttuja ottaa yhteyden käyttäjän pilvipalveluun ja lisää tapahtuman Outlook-kalenteriin.

```

const handleWodTitleChange = (event) => {
  setWodTitle(event.target.value)
}
const handleWodStartChange = (event) => {
  setStartTime(event.target.value)
}
const handleWodEndChange = (event) => {
  setEndTime(event.target.value)
}

```

Treeniä luodessa tarvitaan treenin nimi ja ajankohta. Forms-lomakkeeseen jokaiselle kentälle tarvitaan tapahtumankäsittelijä, joka tallentaa nimen muuttujaan.

```

return (
  <Router>
    <Navbar bg="light" variant="light">
      <Container>
        <Navbar.Brand as={Link} to="/">Crossfit Kirkkopuisto</Navbar.Brand>
        <Nav className="me-auto">
          <Nav.Link as={Link} to="/">Etusivu</Nav.Link>
          <Nav.Link as={Link} to="/wods">Wodit</Nav.Link>
          <Nav.Link as={Link} to="/calendar">Kalenteri</Nav.Link>
          <Nav.Link as={Link} to="/info">Yhteystiedot</Nav.Link>
        </Nav>
        <Navbar.Collapse className="justify-content-end">

```

```

        <Login />
      </Navbar.Collapse>
    </Container>
  </Navbar>

```

App.js palauttaa bootstrap-router-kirjastolla toteutetun valikoidun sisällön renderöimisen. Valmiit komponentit muodostavat navigointi palkin sivulle. React Bootstrap-kirjaston avulla saadaan suoraan React-sovellukselle valmisteltuja komponentteja, jotta sivun saa äkkiä siedettävän näköiseksi. Sivusto on linkitetty ikään kuin toimimaan perinteisen web-sivun tavoin. Linkin jälkeen on polku mihin sivun näkymä siirtyy "/" on kotisivu ja "/calendar" näyttää käyttäjän kalenterin.

3.8 Muut komponentit

Tiedon siirtoon komponenttien välille on parempia tapoja. Sovelluksen monimutkaistuesssa tämä menetelmä ei ole optimaalinen.

```

<Switch>
  <Route path="/wods">
    <Wods wods={wods}/>
  </Route>
  <Route path="/info">
    <Info />
  </Route>
  <Route path="/calendar">
    <UserCalendar wods={wods}
addWod={addWod}handleWodTitleChange={handleWodTitleChange}handleWodStartChange={han
dleWodStartChange}handleWodEndChange={handleWodEndChange}wodTitle={wodTitle}
setWodTitle={setWodTitle}newStartTime={newStartTime} setStartTime={setStartTime}
newEndTime={newEndTime}setEndTime={setEndTime}/>
  </Route>
  <Route path="/">
    <Home />
  </Route>
</Switch>
</Router>

)
}

export default App

```

Käyttäjän kalenteri komponenttiin viedään paljon tietoa liittyen kalenteriin lisäämiseen. Sivulla näytettävää sisältöä säädellään Router-komponentin avulla. Route-komponentin sisään

kirjoitetaan polku ja näytettävä itse kasattu komponentti. Lopussa funktio App viedään eteenpäin. App funktion sisältää sovelluksen koko toiminnallisuuden.

Keskeisiä toimintoja muista komponenteista App.js tiedoston ulkopuolelta. Toimintaa on jaettu eri tiedostoihin koodin ylläpitämisen kannalta.

```
<Table striped bordered hover>
  <thead>
    <tr>
      <th>WODI</th>
      <th></th>
    </tr>
  </thead>
  <tbody>{props.wods.map(wod =>
    <tr key={wod.id}>
      <td>{wod.subject}</td>
      <td></td>
    </tr>)}
  </tbody>
</Table>
```

DailyWods funktio palauttaa päivän treenit map-funktiota käyttäen. Jokaiselle <tr> tagille täytyy määritellä oma avain, johon käytetään treenin id:tä.

```
<Form.Group className="mb-3" controlId="formGroupEmail">
  <Form.Label>WOD start time</Form.Label>
  <Form.Control
    type="datetime-local"
    onChange={props.handleWodStartChange}
    value={props.newStartTime}
  />
</Form.Group>
```

Tilan muuttuessa handleWodStartChange tapahtumankäsittelijä.

```
import axios from 'axios'
const baseUrl = 'http://localhost:3001/api/wods'

const getAll = () => {
  return axios.get(baseUrl)
}
const getOne = (id) => {
  return axios.get(`${baseUrl}/${id}`)
}
```

```
const create = Wod => {
  return axios.post(baseUrl, Wod)
}

const update = (id, Wod) => {
  return axios.put(`${baseUrl}/${id}`, Wod)
}

export default {
  getAll: getAll,
  create: create,
  update: update,
  getOne: getOne,
}
```

Sovelluksen tekemät REST-api kyselyt on eriytetty omaan tiedostoon, jossa käytetään axios-kirjastoa yksinkertaistamaan pyyntöjen tekemistä. Tallennetaan muuttuun rajapinnan osoite. Tiedosto lopuksi määrittelee mitä funktioita sovelluksesta on mahdollista ottaa käyttöön.

```
import wodService from './services/wods'
```

Kyselyt otetaan käyttöön wodService-muuttujan avulla index.js tiedostossa.

3.9 Microsoft Graph

Sovellus rekisteröidään Azure pilvipalvelussa, joka muodostaa clientID:n sovellukselle.

Sovelluksen oikeudet Graph-rajapinnasta. (kuva 10, "Api permissions", 2022)

Dashboard > Default Directory | App registrations > Crossfit Kirkkokuisto

Crossfit Kirkkokuisto | API permissions

Search (Ctrl+/) Refresh Got feedback?

Overview

Quickstart

Integration assistant

Manage

Branding & properties

Authentication

Certificates & secrets

Token configuration

API permissions

Expose an API

App roles

Owners

Roles and administrators

Manifest

Starting November 9th, 2020 end users will no longer be able to grant consent to newly registered multitenant apps without verified publishers. [Add MPN ID to verify, publish](#)

The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may no longer be used. [Learn more](#)

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission ✓ Grant admin consent for Default Directory

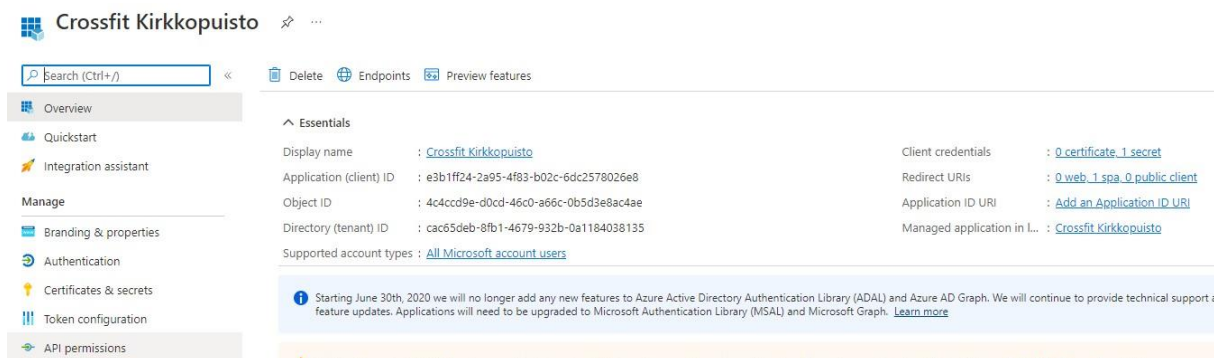
| API / Permissions name | Type | Description | Admin consent required | Status |
|------------------------|-----------|------------------------------------|------------------------|--------|
| Microsoft Graph (2) | | | | |
| Calendars.ReadWrite | Delegated | Have full access to user calendars | No | |
| User.Read | Delegated | Sign in and read user profile | No | |

Otetaan asennetut Microsoftin JavaScript kirjastot käyttöön import komentoilla.

```
import {Providers} from '@microsoft/mgt-element';
import {Msal2Provider} from '@microsoft/mgt-msal2-provider';

Providers.globalProvider = new Msal2Provider({
  clientId: "tähän oma ClientID Azuresta sovelluksen rekisteröinnin jälkeen",
  scopes: ['calendars.readwrite', 'user.read', 'openid', 'profile',
'people.read', 'user.readbasic.all']
});
```

Azure-portaalissa sovelluksen Overview-välilehdellä löytyy sovelluksen tiedot ja Client ID. (kuva 11, ” App registrations”, 2022)



Scopes taulukkoon määritellään mitä tietoja käyttäjän tietoja Microsoft Graph palvelu saa ottaa sovelluksessa käyttöön.

```
import {Agenda} from '@microsoft/mgt-react';
```

```
<Col>
  <Agenda/>
</Col>
```

Microsoftin valmiiksi muodostama komponentti, jolla voidaan renderöidä kalenteri.

```
import ReactDOM from 'react-dom'
import App from './App.js'
```

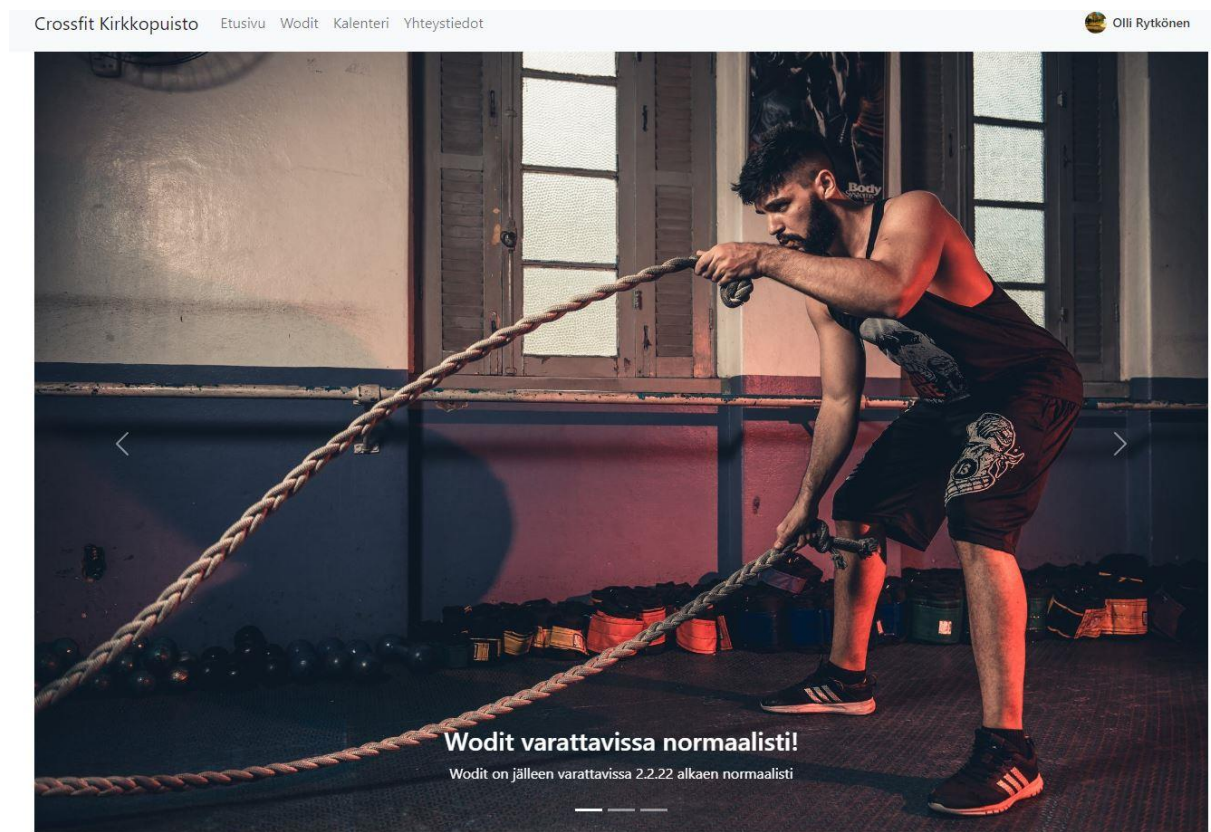
```
ReactDOM.render(<App />, document.getElementById('root'))
```

Yllä oleva koodi on kokonaisuudessaan index.js tiedosto. App funktio vie index.js tiedostoon jossa React.DOM kirjaston avulla vie sovellus HTML-tiedostoon.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Crossfit Kirkkopuisto</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Index.html tiedosto ja perinteiset HTML-dokumentin määrittelyt. Body-tagien sisälle <div> joka renderöi sovelluksen web-sivulle.

Sovelluksen etusivu. (kuva 12)




Sovelluksen etusivulla olisi kolme tuoreinta uutista salin sisäiseen viestintään. Oikeasta yläkulmasta kirjaudutaan sisään sovellukseen.

Wodit välilehti. (kuva 13)

Crossfit Kirkkokuisto
Etusivu
Wodit
Kalenteri
Yhteystiedot
Olli Rytönen

Nex benchmark



Cindy

What is CrossFit Cindy? Cindy is a benchmark workout that consist of 5 pull-ups, 10 push-ups and 15 squats for as many rounds possible (AMRAP) in 20 minutes.

Edellinen tuloksesi

Daily results

Lisää päivän tuloksesi

Kommentti

☐ Only to my private

Submit

Daily Wods

WODI

Näytä tulokset

Olli Rytönen 11 mins ago

Result: **457**
Hyvin kulki tänään! :)

Sauli Niinistö 11 mins ago

Result: **427**
Rx

Wodit-sivulla käyttäjä voi lisätä omat tulokset ja katsoa miten muiden treenit ovat kulkeneet. Oikealla puolella olisi päivän ohjatut treenit. Vasemmalla olisi kuukausittain vaihtuva seuraava suurempi treeni.

Outlook-kalenteri merkinnän tekävä välilehti. (kuva 14)

Crossfit Kirkkokuisto
Etusivu
Wodit
Kalenteri
Yhteystiedot
Olli Rytönen

Wods

WODI

Add WOD to calendar

WOD start time

mm/dd/yyyy --:-- --

WOD end time

mm/dd/yyyy --:-- --

Wod Name

Submit

Your Outlook Calendar

Vasemmalla puolella olisi päivän treeni, josta lisätä treeni suoraan Outlook kalenteriin. Tieto ei siirtynyt MongoDB-tietokannasta suoraan Outlook-kalenteriin, joten merkintä piti lisätä suoraan Outlook-kalenteriin.

Treenin luominen. Aikavyöhyke ei ole kunnossa. (kuva 15)

Add WOD to calendar

WOD start time

08/13/2022 01:00 PM

WOD end time

08/13/2022 02:00 PM

Wod Name

Penkkiä

Submit

Your Outlook Calendar

Treeni Outlook kalenterissa. (kuva 16)

Add WOD to calendar

WOD start time

mm/dd/yyyy --:-- --

WOD end time

mm/dd/yyyy --:-- --

Wod Name

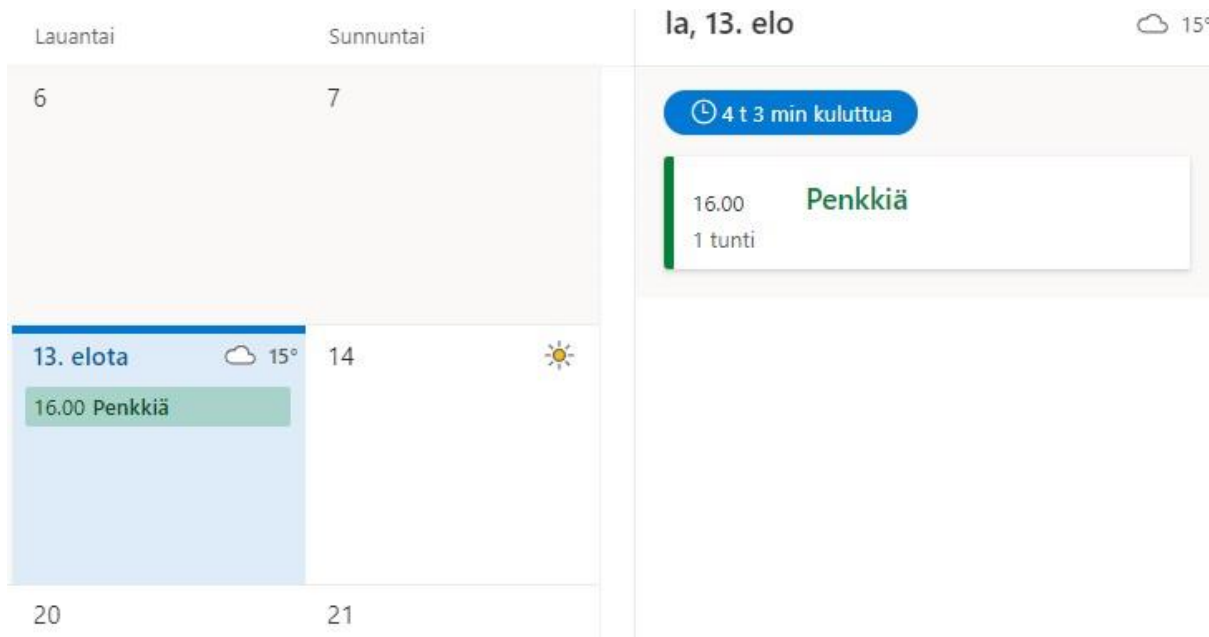
Submit

Your Outlook Calendar

7:00 PM - 8:00 PM

Penkkiä

Outlook kalenterin näkymä. (kuva 17)



4 Johtopäätökset

Sovelluksessa päästiin tavoitteeseen luomalla SPA-sovellus ja liittämällä siihen Microsoft Graph-palvelu, mutta käyttöliittymä ja tietokantojen kommunikointi keskenään jäi kesken.

Sovelluksessa on parannettavaa runsaasti. Näkyvin virhe treeniä lisättäessä on aikavyöhyke ja sekava käyttöliittymä. Sovelluksen toiminnot pitäisi tiivistää yhteen sivuun, eikä jakaa toimintoja monelle välilehdelle. Wodit-välilehden oli tarkoitus olla pääkäyttäjän näkymä, missä voidaan luoda treenejä koko salin käyttöön. Treenit siirtyisivät MongoDB-tietokantaan. Kalenteri-välilehdellä olisi valmiina treenejä, joista käyttäjä voi valita treenin ja lisätä merkinnän Outlook-kalenteriin. Treenin sai kyllä luotua MongoDB-tietokantaan, mutta treenin siirtäminen MongoDB-tietokannasta Outlook-kalenteriin ei onnistunut. Muutin treenin lisäämisen suoraan sovelluksesta Outlook-kalenteriin. Toimintoja Graph-palvelusta voisi ottaa käyttöön myös enemmän, esimerkiksi Teams-ryhmän ja sen myötä keskustelumahdollisuuden. Virheiden käsittely ja yleinen testaaminen rajattiin työstä pois, mutta niiden lisääminen olisi seuraava vaihe työssä.

Onnistumiseen lukeutuu se, että treenin sai lisättyä Outlook-kalenteriin ja MongdoDB-tietokantaan. Koodin kirjoittaminen onnistui myös ilman ohjevideoita ja keskityin virallisiin dokumentaatioihin.

Tästä on hyvä jatkaa kehittämistä ja viimeistellä sovellus sekä tutkia, mitä muita mahdollisuuksia Graph-palvelu tarjoaa. Opinnäytetyön ohjauksesta oli apua motivoimaan ja hahmottamaan kokonaisuutta. Suuri apu vuosien varrella on ollut Helsingin yliopiston kaikille avoin Full Stack Open kurssi (Luukkainen, M. H. (2022)). Opinnäytetyöstä itsestään jää ristiriitainen olo, koska tavallaan tavoite täyttyi, mutta vielä enemmän aikaa käyttämällä työstä saisi hiottua uskottavan treenisovelluksen.

5 Lähteet

About Node.js - Node. (2022). <https://nodejs.org/en/about/>

About SQLite - SQLite. (n. d.). <https://www.sqlite.org/about.html>

App registrations. - Microsoft. (2022).

https://portal.azure.com/#view/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/~RegisteredApps

Api permissions. - Microsoft. (2022).

https://portal.azure.com/#view/Microsoft_AAD_RegisteredApps/ApplicationMenuBlade~/CallAnAPI/appId/e3b1ff24-2a95-4f83-b02c-6dc2578026e8/isMSAApp~/false

Choose Between Traditional Web Apps And Single Page App (SPAs) - (2022).

<https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>

Graph Explorer [kuva]. - Microsoft. (2022). <https://developer.microsoft.com/en-us/graph/graph-explorer>

How the sample works [kuva]. - Microsoft. (2022). <https://docs.microsoft.com/en-us/azure/active-directory/develop/single-page-app-quickstart?pivots=devlang-angular>

Syväsukellus moderniin websovelluskehitykseen - Luukkainen, M. H. (2022).

<https://fullstackopen.com/>

Roles in OAuth 2.0, Tokens, App registration, Endpoints. - (2022). <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-v2-protocols>

Microsoft identity platform [kuva]. – (2022) <https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-overview>

Microsoft 365 Platform [kuva]. - Microsoft. (2022) <https://docs.microsoft.com/en-us/graph/overview>

Microsoft Graph [kuva]. - Microsoft. (2022). <https://docs.microsoft.com/en-us/graph/overview>

npmtrends. - (2022) . <https://npmtrends.com/@angular/core-vs-angular-vs-react-vs-vue>

Schaefer, L. (n. d.). What is NoSQL <https://www.mongodb.com/nosql-explained>

StateofJS. [kuva]. -(2022 -a). <https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/>

StateofJS. [kuva]. - (2022 -b). <https://2021.stateofjs.com/en-US/libraries/back-end-frameworks>

Top IDE index - (2022) <https://pypl.github.io/IDE.html>

VueJS – Overview - (2022). https://www.tutorialspoint.com/vuejs/vuejs_overview.htm

What Is a Database? - Oracle. (2022). <https://www.oracle.com/database/what-is-database/>

What is PostgreSQL – (2019) <https://www.ibm.com/cloud/learn/postgresql>