VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Niko Lehtimäki

CRM-JÄRJESTELMÄN RAKENTAMINEN SHAREPOINT-YMPÄRISTÖÖN

Case Isännöinti Mäkinen Oy

Liiketalous
2022

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Niko Lehtimäki
Opinnäytetyön nimi	CRM-järjestelmän rakentaminen SharePoint ympäristöön Case Isännöinti Mäkinen Oy
Vuosi	2022
Kieli	Suomi
Sivumäärä	54
Ohjaaja	Päivi Rajala

Tämä opinnäytetyö käsittelee CRM-järjestelmää, joka rakennetaan SharePoint-ympäristöön. Työssä tuodaan esille keskeisiä tekniikoita, joita tarvitaan järjestelmän rakentamiseen ja niiden yhdistämistä yhdeksi toimivaksi kokonaisuudeksi. Työn toimeksiantajana toimi Pilvilampi Software Oy.

Opinnäytetyön tavoite on rakentaa järjestelmä käyttäen Pilvilammen Sopivin 365-sovellusalustaa pohjana. Järjestelmällä helpotetaan työnkulkua sekä mahdollistetaan yritystä koskevien tietojen vaivaton säilytys, muokkaus ja lisäys. Järjestelmän avulla yrityksen tiedot ovat aina turvassa ja helposti saatavilla yhdestä paikasta.

Opinnäytetyön teoriatausta koostuu pääosin sivuston toteuttamiseen vaadittavista teknikoista. Tässä projektissa käytettävät tekniikat ovat SharePoint, Power Automate ja AngularJS. Opinnäytetyö on empiirinen, laadullinen tapaustutkimus, joka on toteutettu projektina.

Opinnäytetyön lopputuloksena syntyi vaatimusmäärittelyä vastaava ajanmukainen ja käyttäjäystävällinen sivusto, joka on suunniteltu, toteutettu, testattu ja luovutettu asiakkaalle. Asiakas on ottanut sivuston käyttöön ja antanut palautetta projektin lopputuloksesta. Projektia jatketaan vielä korjaamalla virheellisiä osia ja tekemällä parannuksia sivuston käytettävyyteen sitä mukaa, kun asiakas testaa sivustoa.

Avainsanat SharePoint, Power Automate, AngularJS, Ohjelmistokehitys

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittely

ABSTRACT

Author	Niko Lehtimäki
Title	Building a CRM System into a SharePoint Environment Case Isännöinti Mäkinen Oy
Year	2022
Language	Finnish
Pages	54
Name of Supervisor	Päivi Rajala

This thesis studied a CRM system that was built on SharePoint environment. During the thesis process all essential concepts needed to build the system and how to combine them as one were studied. The work was commissioned by Pilvilampi Software Oy.

The objective of this thesis was to build a system using Pilvilampi's Sopivin 365 platform as a base. The system is used to ease workflow and make it so that all the company's information is easy to store, modify, and new information can be added. With the system all the company's information is kept safe and easily accessible in one place.

The theoretical section of this thesis is mostly composed of techniques required to implement a site. The techniques in this project are SharePoint, Power Automate and AngularJS. This thesis is an empirical, qualitative case study, that was carried out as a project.

The final product of this thesis was a site that is user-friendly and up to date while also corresponding with the required specifications. The site was designed, implemented, tested, and handed over to the customer as the end of the process. The customer has put the site to use and given feedback regarding the result of this project. The project will be continued by patching incorrect sections and by improving the site's usability as the customer tests the site.

Keywords SharePoint, Power Automate, AngularJS, Software Development

SISÄLLYS

KUVALUETTELO

1 JOHDANTO

Opinnäytetyön tarkoituksena on uuden CRM-järjestelmän rakentaminen SharePoint ympäristöön ja rakentamisen havainnollistaminen. Järjestelmän tarkoitus on helpottaa ja automatisoida prosesseihin liittyviä työvirtoja ja -tehtäviä ja kehittää yrityksen työprosesseja. Kyseisiä työprosesseja ovat esimerkiksi dokumenttien generointi, asiakkuuksienhallinta ja myyntityö.

Tämä opinnäytetyö on empiirinen laadullinen tapaustutkimus, joka toteutetaan projektina. Projektissa rakennetaan asiakkaalle CRM-järjestelmä SharePoint-ympäristöön. Aineistonhankinta perustuu projektin havainnointiin ja tekniikoiden tarkasteluun.

Opinnäytetyön teoriaosuudessa selvitetään oleellisia tekniikoita ja käsitteitä, kuten SharePoint, Power Automate ja AngularJS, jotka ovat tärkeitä osia järjestelmän rakentamisessa. Teoriaosuudessa käsitellään näiden tekniikoiden sisältöä ja niiden tarvetta selitetään ennen projektin toteutusta, jossa niiden käyttötarkoitus esitellään tarkemmin.

Projektin toteutusosioissa selvitetään, miten SharePointia, Power Automatea ja AngularJS:ää käytetään projektin rakentamisessa. Tarkoituksena on myös osoittaa, miten nämä kolme tekniikkaa saadaan toimimaan yhdessä, mitä niiden toiminta vaatii ja miten ne vaikuttavat toisiinsa.

Projektin toimeksiantajana toimii Pilvilampi Software Oy. Pilvilampi Software Oy on vaasalainen oivaltava digitaalisen työn edistäjä ja vahva Microsoft O365-asiantuntija, joka kehittää Sopivin O365 -liiketoiminta-alustaa monipuolisiin tiedonhallinnan ratkaisuihin. Järjestelmän käyttöönotossa asiakkaalta vaaditaan vain Microsoft 365 -ympäristö, jota Sopivin 365 hyödyntää. Pilvilampi Software Oy:n markkina-alue kattaa koko Suomen. Projekteja on tehty useisiin yrityksiin, useisiin eri kaupunkeihin, kuten esimerkiksi Tampereelle, Kauhavalle ja Vaasaan.

2 SHAREPOINT

SharePoint on maksullinen Microsoft 365-ympäristön lisäosa, joka tarjoaa työkaluja datan sekä dokumenttien jakamiseen yhtiön työntekijöiden, asiakkaiden ja yhteistyökumppaneiden välillä. SharePointia käytetään näiden lisäksi myös erilaisten prosessien sekä toimenpiteiden hallintaan ja sitä voidaan pitää myös työnkulkua organisoivana työkaluna. SharePointin päälle on myös mahdollista konfiguroida omanlaisensa sovellus, jota voidaan käyttää esimerkiksi asiakkuuksien, kiinteistöjen ja inventaarion hallinnassa.

SharePoint tukee käyttäjiä ja käyttäjäryhmiä. Käyttäjille ja ryhmille voidaan myöntää lupa SharePoint sivuston eri osiin tai heiltä voidaan estää pääsy niihin. Asianomaisille käyttäjille voidaan myöntää oikeudet sivuston sisällön muokkaukseen (Alexander 2016, 126.)

SharePoint pitää automaattisesti yllä versiohistoriaa kaikesta datasta ja objekteista. Muutokset voidaan siis palauttaa aikaisempaan tilaan käytännössä, milloin vain halutaan. Mahdollisuus muutosten palautukseen voidaan myöntää yksittäisille käyttäjille, joten DBA-tuki (Database administrator) ei ole tällöin tarpeellinen (Alexander 2016, 126.)

Poistetut datat ja objektit säilyvät SharePointin roskakorissa, josta ne voidaan helposti palauttaa tarpeen mukaan. SharePoint tukee kumoa-toimintoa kaikelle datalle (Alexander 2016, 126.)

Käyttäjiä ja ryhmiä voidaan hälyttää lähettämällä sähköpostiviestejä aina kuin tiettyä dokumenttia lisätään, poistetaan tai muutetaan SharePointissa. Asianmukaisella oikeudella käyttäjät voivat itse hallita omia hälytyksiään (Alexander 2016, 126.)

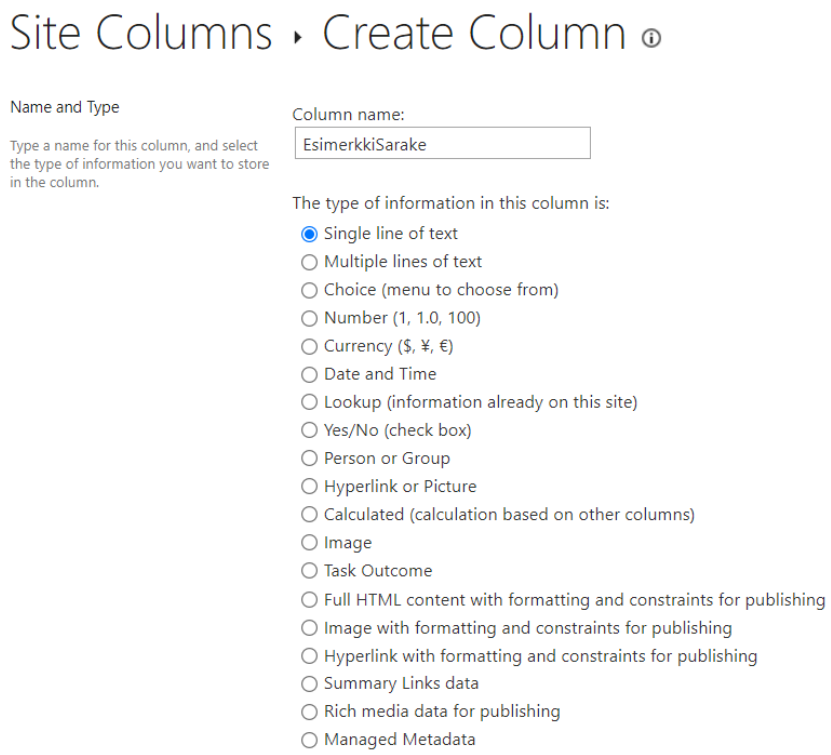
SharePoint sivustojen ylläpito on yleensä käyttäjien vastuulla, eikä siihen tarvita IT-puolen apua. SharePointin sivustot eivät ole yhtä joustavia, kuin normaalit verkkosivustot, mutta SharePoint-ohjelmistokehittäjä voi lisätä tai poistaa toi-

mintoja sivuilta. Hän pystyy myös vaikuttamaan sivuston fontteihin, otsikoihin, väreihin ja muihin sivuston piirteisiin, kuten esimerkiksi alasivujen ja listojen muodostumiseen (Alexander 2016, 126.)

2.1 Sarakkeet

Sarakkeiden (Columns) avulla voidaan luoda kuvaavia näkymiä luettelon tai kirjaston kohteista. Sarakkeita käyttämällä voidaan lajitella, ryhmitellä ja suodattaa kohteita. Sarakkeilla voidaan myös määritellä, mitkä tiedot kohteesta on annettava, kun lisätään tietoa luetteloon tai kirjastoon. Tietyt sarakkeet, kuten otsikko ja muokkaaja luodaan aina automaattisesti uudelle luettelolle tai kirjastolle. Lisäsarakkeita voidaan luoda omien tarpeiden mukaan. Luodut sarakkeet määrittävät, mitä kenttiä ja asetuksia näytetään lomakkeessa ja mitä sarakkeita voidaan lisätä luettelon tai kirjaston näkymiin. (Microsoft 2021e.)

Sarakkeen tyyppi määritellään, kun sarake luodaan (Kuva 1). On myös mahdollista muokata aiemmin luodun sarakkeen saraketyyppiä, mutta tämä riippuu sarakkeeseen jo tallennettujen tietojen tyypistä ja määrästä. Aikaisemmin luodun sarakkeen saraketyypin vaihtaminen voi aiheuttaa virheen riippuen aikaisemmin tallennetusta tiedosta. Tämän takia on tärkeä päättää tallennettava saraketyyppi ennen kuin sarake luodaan. (Microsoft 2021e.)



Kuva 1. Uuden SharePoint sarakkeen luonti.

2.1.1 Yksi tekstirivi

Yhden tekstirivin sarake (Single line of text) voi pitää sisällään enintään 255 merkkiä. Sitä käytetään keräämään ja näyttämään pientä määrää muotoilematonta tekstiä yhdellä rivillä, näitä voivat esimerkiksi olla:

- Pelkkä teksti, kuten henkilöiden nimet tai osastojen nimet
- Tekstin ja numeroiden yhdistelmät, kuten osoitteet tai tilinumerot
- Numerot, joita ei käytetä laskemiseen, kuten puhelinnumerot tai postinumerot.

Yhden tekstirivin saraketta voidaan räätälöidä asettamalla raja sille, kuinka monta merkkiä sarake saa sisältää. Esimerkiksi jos sarakkeen halutaan sisältävän viisi merkkiä pitkän työntekijätunnuksen, voidaan tätä toimintoa käyttää varmistamaan, että kaikkien työntekijöiden tunnukset pysyvät oikean pituisina. (Microsoft 2021e.)

Yhden tekstirivin sarakkeelle voidaan myös asettaa automaattisesti näkymään tietty oletusarvo, kun uusi kohde lisätään ja antaa tarvittaessa käyttäjien lisätä muuta tekstiä, jos sille on tarve. Määrittämällä oletusarvon, käyttäjät voivat syöttää tietoja nopeammin hyväksymällä oletusarvon, ellei sitä tarvitse muuttaa. Oletusarvo voi olla määritelty teksti tai laskutoimituksen tulos. (Microsoft 2021e.)

2.1.2 Haku

Hakusarake (Lookup) on saraketyyppi, joka mahdollistaa sarakkeen yhdistämisen toiseen listan tai kirjaston sarakkeeseen, joka on jo luotu omalle SharePoint-sivustolle. Hakusarakkeella voi hakea tietoa tästä toisesta sarakkeesta, vaikka se onkin toisella listalla (Zelfond 2020). Esimerkkinä on Tilaukset-luettelo, jossa linkitetään Asiakkaat-luetteloon hakusarake, jotta nähdään tilauksen tehnyt asiakas.

Hakusarakkeeseen voidaan asettaa poistorajoitus käyttäen, joko poiston rajoitustoimintoa (restrict delete) tai johdannaispoistoa (cascade delete). Poiston rajoitustoiminto estää kohteen poistamisen, jos siihen liitettäviä lähdeluettelon kohteita on vielä olemassa. Johdannaispoistolla varmistetaan, että kaikki toisiinsa liittyvät kohteet poistetaan samassa tietokantatapahtumassa. (Microsoft 2021c.)

2.1.3 Useita tekstirivejä

Usean tekstirivin saraketta (Multiple lines of text) käytetään keräämään ja näyttämään muotoiltua tekstiä tai pitkää tekstiä ja numeroita vähintään yhdellä rivillä. Usean tekstirivin sarake voi pitää sisällään 63 999 merkkiä ja näytettävien tekstirivien määrän voi määritellä haluamakseen. Tämän tyypin sarake näyttää koko tekstin, kun saraketta tarkastellaan listassa tai kirjastossa. (Microsoft 2021e.)

Usean tekstirivin saraketta voidaan räätälöidä haluamakseen määrittelemällä näyttörajoituksen ja tekstin muotoilun. Näyttörajoituksella rajataan, kuinka monta riviä ilmestyy näkyviin, kun käyttäjä antaa tietoja kohteesta. Tekstin muotoilul-

la määritellään voivatko käyttäjät käyttää erilaisia tekstinmuotoiluja, kuten lihavointia, kursivointia tai värejä. (Microsoft 2021e.)

2.1.4 Numero ja valuutta

Numerosaraketta (Number) käytettään numeeristen arvojen tallentamiseen, jotka eivät ole raha-arvoja. Numerosaraketta voidaan käyttää matemaattisia laskelmia varten, jotka eivät ole talouslaskelmia tai joilta ei vaadita suurta tarkkuutta. Numerosaraketta voidaan mukauttaa seuraavilla tavoilla:

- Vähimmäis- ja enimmäisarvojen määritys
- Desimaalien lisäys
- Oletusarvon määrittely
- Luvun muotoilu prosentiksi

Valuuttasaraketta (Currency) käytetään raha-arvojen tallentamiseen. Valuuttasarakkeeseen tallennettavat numerotiedot mahdollistavat talouslaskelmat ja laskelmat, joissa ei haluta pyöristettyjä lukuja. Numerosarakkeeseen verrattuna valuuttasarake on tarkempi. Valuuttasarakkeen mukautus toimii samalla tavalla kuin numerosarakkeen, mutta luvun prosentiksi muotoilun sijaan on mahdollista valita valuuttatyyppi. (Microsoft 2021e.)

2.1.5 Henkilö tai ryhmä

Henkilö- tai ryhmäsarake (Person or Group) on luettelo, josta käyttäjä voi valita haluamansa henkilön tai ryhmän kohdetta muokatessa tai lisätessä. Luettelon sisältö riippuu siitä, miten hakemistopalvelut ja SharePoint-ryhmät on määritelty sivustoa varten. Henkilö tai ryhmä -saraketta voidaan mukauttaa sallimalla useita valintoja ja sisällyttämällä tai jättämällä pois käyttäjäryhmiä. (Microsoft 2021e.)

Henkilöitä voidaan paikantaa käyttämällä henkilön nimeä tai sähköpostiosoitetta. Sarake etsii automaattisesti sopivia nimiä saatavilla olevista arvoista käyttäjän kirjoittaessa sarakkeeseen.

2.1.6 Päivämäärä ja aika

Päivämäärä- ja aikasaraketta (Date and Time) käytettään tallentamaan kalenteripäivämääriä. Päivämäärän muoto vaihtelee sivuston aluekohtaisista asetuksista riippuen. Järjestelmänvalvoja voi lisätä halutun alueen tuen sivustolle. (Microsoft 2021e.)

Päivämäärä- ja aikasaraketta voidaan räätälöidä, siten että näytetään pelkkä päivämäärä tai päivämäärä ja aika yhdessä. Sarakkeelle voidaan asettaa myös oletusarvo aina kuin käyttäjä lisää uuden kohteen. Oletusarvolla voidaan nopeuttaa tietojen lisäystä luettelolle mutta se on myös tarvittaessa muokattavissa. (Microsoft 2021e.)

2.1.7 Valinta

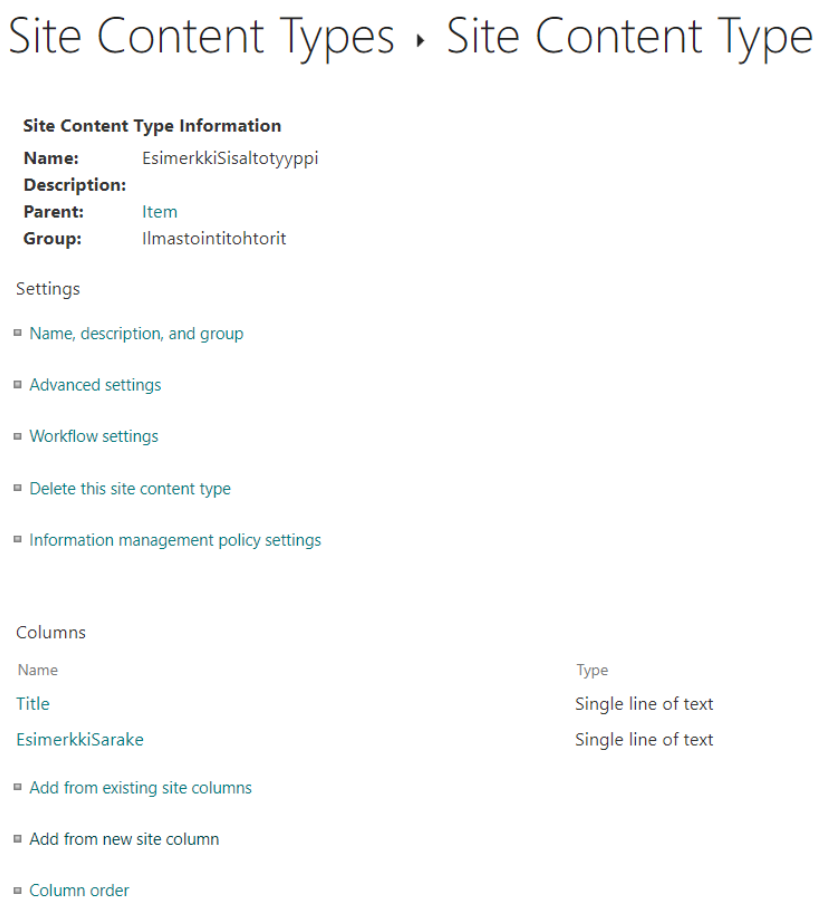
Valintasarakkeen (Choice) avulla käyttäjät voivat valita, jonkin ennalta määritetyn vaihtoehdon. Tämä saraketyyppi sopii tapauksiin, joissa halutaan varmistaa, että kaikki tiedot sarakkeessa ovat yhdenmukaisia, koska sillä voidaan rajoittaa sarakkeeseen tallennettavia arvoja. Käyttäjät voivat tehdä valintansa, joko avattavasta valikosta tai valintapainikkeista. (Microsoft 2021e.)

Valintasaraketta voidaan mukauttaa määrittämällä vaihtoehtojen luettelo, ottamalla käyttöön mukautettuja lisävaihtoehtoja ja näyttämällä oletusarvon. Mukautetuilla lisävaihtoehdoilla käyttäjän on mahdollista lisätä arvoja manuaalisesti, jos arvoja ei luettelossa vielä ole. (Microsoft 2021e.)

2.2 Sisältötyypit

Sisältötyypit (Content Types) ovat kokoelma sarakkeita, jotka ovat ryhmitelty yhteen, tietyllä tarkoituksella (kuva 2). Esimerkiksi voidaan luoda sisältötyyppi nimeltään ”Laskut”, joka voidaan upottaa jokaiselle luettelolle tai kirjastolle, joka tarvitsee kyseisiä tietoja. Aina kun tarvitaan kokoelma tiettyjä sarakkeita, voidaan ne lisätä luettelolle tai kirjastolle yhdellä napin painalluksella, sen sijaan,

että luodaan ja lisätään, jokainen sarake yksi kerrallaan. SharePointissa, voidaan myös käyttää useita sisältötyyppejä luettelon tai kirjaston sisällä, eli voidaan käyttää eri kokoelmaa sarakkeita, eri luettelokohteilla tai dokumenteilla, jotka sijaitsevat samassa luettelossa tai kirjastossa. (Niaulin 2014.)

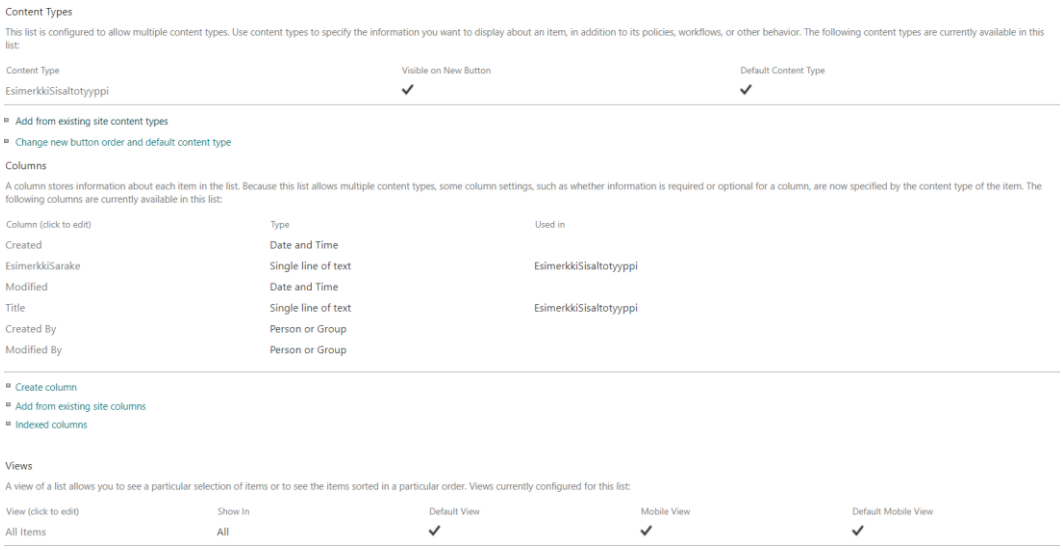


Kuva 2. Sisältötyyppi, jolle on lisätty aikaisemmin luotu sarake.

2.3 Luettelot

Luettelo (List) on kokoelma tietoja, joka mahdollistaa organisaatiolle joustavan tavan organisoida informaatiota. Tiedot tallentuvat SharePoint sivustolle riveihin ja sarakkeisiin. Sarakkeiden nimet ja tyypit määrittelevät minkä tyyppinen luettelo on kyseessä. Luettelot voivat sisältää useita erilaisia tietoja, kuten teksti, valuutta tai päivämäärä. Jokainen uusi kohde lisää uuden rivin luetteloon. Luettelolle voidaan myös luoda näkymiä, joiden avulla tietoja tuodaan esille tehok-

kaammin. Luetteloa on myös mahdollista lajitella, ryhmitellä ja suodattaa sen sisältämän tiedon mukaan. (Reinders 2021.)



Kuva 3. Luettelon asetukset.

Kuva 3 havainnollistaa SharePoint luetteloa, jolle on lisätty aikaisemmin luotu sisältötyyppi. Sisältötyyppi pitää sisällään aikaisemmin luodun sarakkeen, jota voi nyt käyttää luettelolla tietojen tallentamiseen ja näkymien luontiin.

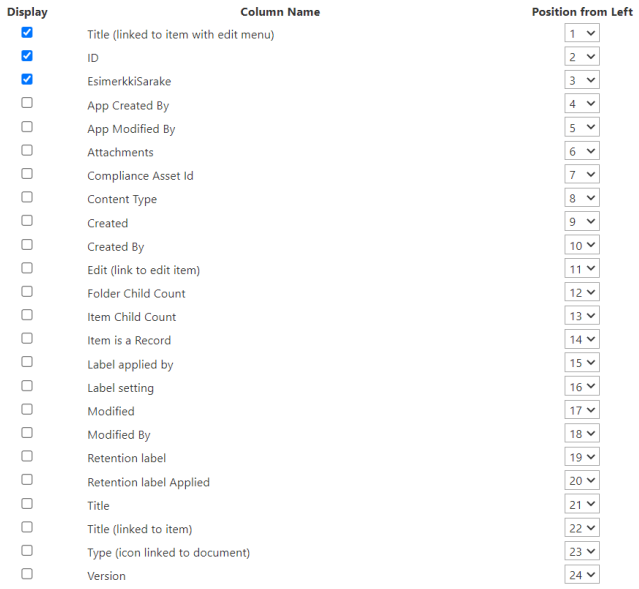
2.4 Tiedostokirjastot

Tiedostokirjasto (Document library) tarjoaa turvallisen paikan tiedostojen säilytykseen. Se on paikka, josta kaikkien käyttäjien on helppo löytää dokumentteja, työstää niitä samaan aikaan ja päästä niihin käsiksi useilta eri laitteilta kerrallaan. Tiedostokirjastoja on mahdollista luoda tiettyjen projektien alle ja siirtää kaikki projektiin liittyvät dokumentit yhteen paikkaan. Tiedostojen siirtäminen tiedostokirjaston kansioiden välillä tapahtuu drag and drop-metodilla. (Joseph 2021.)

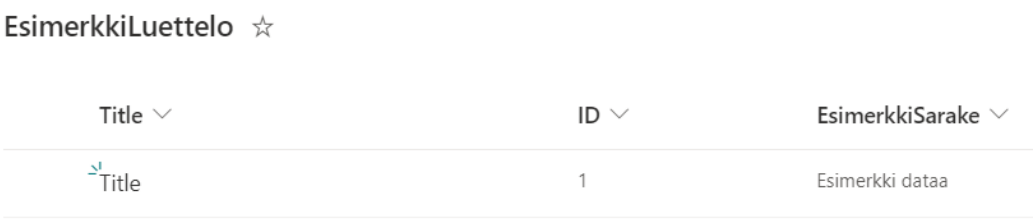
Vaikka luettelot ja kirjastot sisältävät samantapaisia piirteitä, niissä on silti eroja. Suurin ero näiden välillä on se, että kirjastossa, jokainen rivi on yhteydessä dokumenttiin. Tämä dokumentti voi olla minkälainen tahansa, kuten esimerkiksi Office-dokumentti, kuva tai web-sivu. Office-dokumenttien käytössä on se etu, että SharePointilla on yhteys Officen työkaluihin. (tutorialspoint 2021b.)

2.5 Näkymät

Näkymiä (Views) käytettään luetteloiden ja kirjastojen järjestämiseen, kun halutaan näyttää tärkeimpiä kohteita tai lisätä suodatusta ja lajittelua. Näkymästä voidaan esimerkiksi poistaa näkyvistä turhia sarakkeita, joiden sisältämä tieto ei ole organisaatiolle tärkeää (kuva 4). Näkymiä voidaan myös käyttää vähentämään tiedon määrää, jota Power Automatella haetaan. Käyttäjä voi luoda itselleen oman näkymän, jota muut eivät näe tai julkisen näkymän kaikille, jos käyttäjällä on tarvittavat käyttöoikeudet. (Microsoft 2021b.)



Kuva 4. Näkymän luominen.



Kuva 5. Luotu näkymä luettelolla.

Kuva 5 esittää aikaisemmalla konfiguroinnilla (kuva 4) luettelolle luotua näkymää. Tälle kyseisellä näkymällä on jätetty näkyviin vain sarakkeet Title, ID ja EsimerkkiSarake.

2.6 Metatiedot

Metatiedoilla (Metadata) tarkoitetaan tietoa tiedosta. SharePointissa metatiedot ovat lisäinformaatiota tiedostoista, kuten esimerkiksi tekijä, tiedostonimi, luonti päivä, sisältötyyppi tai tiedostokoko, näiden avulla on helpompi paikantaa, palauttaa ja organisoida sisältöä. (Mc Dermid 2017.)

SharePoint käyttää hyväkseen metatietoja yhdessä näkymien kanssa mahdollistamaan käyttäjiä ryhmittelemään, suodattamaan ja järjestämään tietoja, miten he haluavat. Sarakkeita käytetään näyttämään ja hallitsemaan metatietoja luetteloiden ja kirjastojen sisällä. Käyttäjät ja ylläpitäjät voivat valita, mitä sarakkeita näytetään, jotta kaikesta oleellisimmat tiedot ovat esillä. (Mc Dermid 2017.)

Jotkin SharePointin metatietokentät ovat automaattisesti täytettyjä eikä niitä ei ole mahdollista muokata, kun taas muita kenttiä voidaan luoda käyttäjän syötteen mukaan. Standardoidut metatiedot voivat olla erittäin hyödyllisiä mutta mukautetuista metatietokentistä on mahdollista saada vielä enemmän irti organisaatiotasolla. (Mc Dermid 2017.)

3 POWER AUTOMATE

Power Automate, joka aiemmin tunnettiin nimellä Microsoft Flow, on osa Power Platform-nimistä tuoteperhettä. Se on työnkulkua helpottava sovellus, jota voidaan käyttää automatisoimaan prosesseja erilaisia ehtoja ja skenaarioita hyödyntäen. Power Automate sisältää useita Microsoftin ja muiden kehittäjien sovelluksia, joita voidaan käyttää web-pohjaisessa kehityksessä. Power Automatea voidaan käyttää esimerkiksi dokumenttien generoinnissa, Sharepointin dataa sisältävien sähköpostien lähetyksessä ja saapuvien sähköpostien datan lisäyksessä suoraan Sharepointiin.

SharePointissa automatisoidaan prosesseja ja tehtäviä SharePointin ympäristön sisällä. Haittapuolena on, että automatisointia voidaan tehdä pelkästään SharePoint ympäristössä. (Guilmette 2020). Tämän takia Power Automate on loistava ratkaisu Sharepoint-ympäristön ulkopuolisten prosessien automatisointiin.

Power Automate sisältää paikallisen yhteyden satoihin sovelluksiin. Se on myös laajennettava eli siihen voidaan kehittää liitettävyysratkaisuja, jotka toimivat yhteydessä omiin kustomoituihin sovelluksiin. HTTP:n avulla päästään käsiksi melkein minkä tahansa sovelluksen REST-pohjaiseen käyttöliittymään. (Guilmette 2020.)

3.1 REST

REST (Representational State Transfer) tarkoittaa metodia, jolla ollaan vuorovaikutuksessa muiden tietokonejärjestelmien kanssa. REST-pohjaisessa järjestelmässä asiakaslaitteet lähettävät yleensä HTTP-attribuutteja (Hypertext Transfer Protocol) tai toimintoja kohdejärjestelmän URI:lle (Uniform Resource Identifier) datan syöttämiseksi tai hakemiseksi. (Guilmette 2020.)

REST toimii itsenäisesti perustana olevista protokollista, eikä välttämättä ole sidottuna HTTP:hen. Suurin osa REST-toteutuksista, kuitenkin käyttävät HTTP:tä sovellusprotokollana. REST käyttää avointa standardia, eikä sido API tai asiakas-

sovelluksen toteutusta millekään tietylle toteutukselle. REST-verkkopalvelu ja asiakassovellus voivat siis käyttää mitä tahansa ohjelmointikieliä, jotka voivat generoida HTTP-pyynnön ja jäsentää HTTP-vastauksen. (Microsoft 2021f.)

3.2 Flow

Flow tarkoittaa liittimien, käynnistäjien, ehtojen ja toimintojen loogista ryhmitystä, jolla voidaan automatisoida haluttuja toimintoja. Flowt voidaan jakaa seuraaviin kategorioihin:

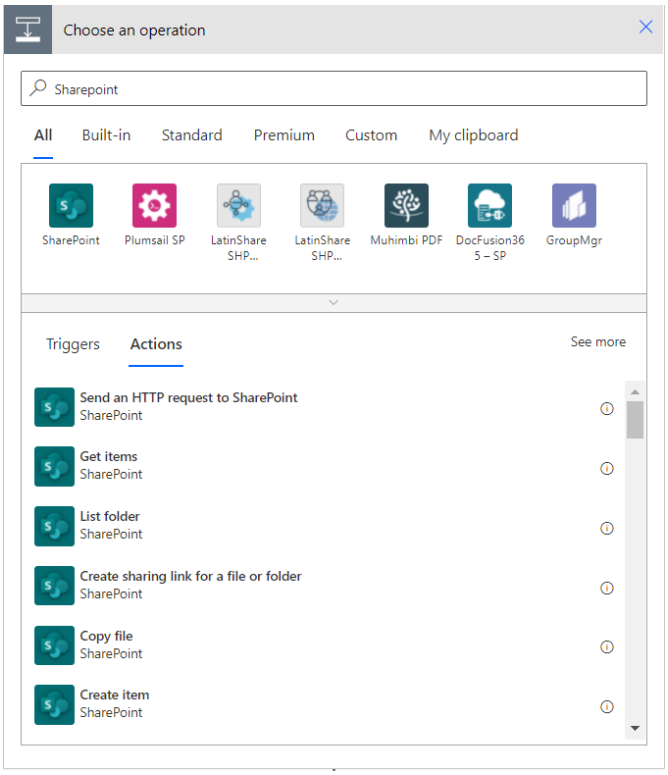
- Automatisoitu
- Painike
- Ajastettu
- Hyväksyntä
- Liiketoimintaprosessi
- UI flowt.

Jokaisella flowlla on erilaisia käyttötarkoituksia, käynnistäjiä ja konfigurointimahdollisuuksia. (Guilmette 2020.)

3.2.1 Liittimet

Liittimet (Connectors) ovat komponentteja, joita käytetään yhtymäkohtana lähde ja tavoite järjestelmien välillä (kuva 6). Liittimet sisältävät tietoa, jota tarvitaan vuorovaikutuksessa applikaatioiden kanssa. Ne voidaan jakaa standardeihin ja maksullisiin liittimiin. Standardit liittimet ovat yleensä mukana kaikissa Power Automate-ratkaisuissa, kun taas maksulliset liittimet sisältävät ylimääräisiä kustannuksia. (Guilmette 2020.)

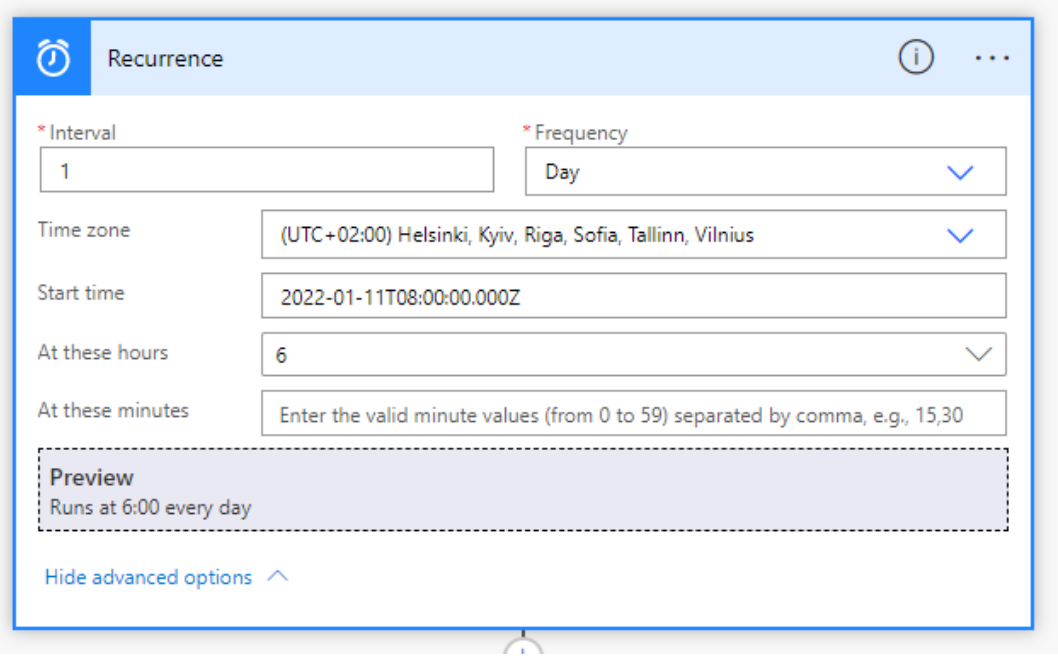
Microsoftin julkaisemat liittimet voivat olla yhteydessä niin ulkoisiin, kuin sisäisiin palveluihin. Ulkoisia palveluita käyttävät liittimet kulkevat läpi Microsoftin vahvistustarkastuksesta, kun ne poistuvat Microsoftin omasta infrastruktuurista. Tällä tavoin varmistetaan, että tieto pysyy turvassa. (Microsoft 2021d.)



Kuva 6. Power Automaten SharePoint-liittimiä.

3.2.2 Käynnistäjät

Käynnistäjät (Triggers) ovat toimintoja, jotka aloittavat flown läpikulun. Ne voidaan jakaa tyypiltään automatisoituihin, manuaalisiin ja ajastettuihin käynnistäjiin. Automaattinen käynnistäjä aloittaa flown, jonkin tietyn tapahtuman mukaan, esimerkiksi sähköpostiin saapuva viesti. Manuaalinen käynnistäjä vaati käyttäjän, joka käynnistää sen, esimerkiksi painamalla nappia. Ajastettu käynnistäjä on aikapohjainen ja käynnistyy aina tietyin väliajoin, esimerkiksi joka päivä aamu kuudelta (Kuva 7). (Guilmette 2020.)



Kuva 7. Aikapohjainen käynnistäjä.

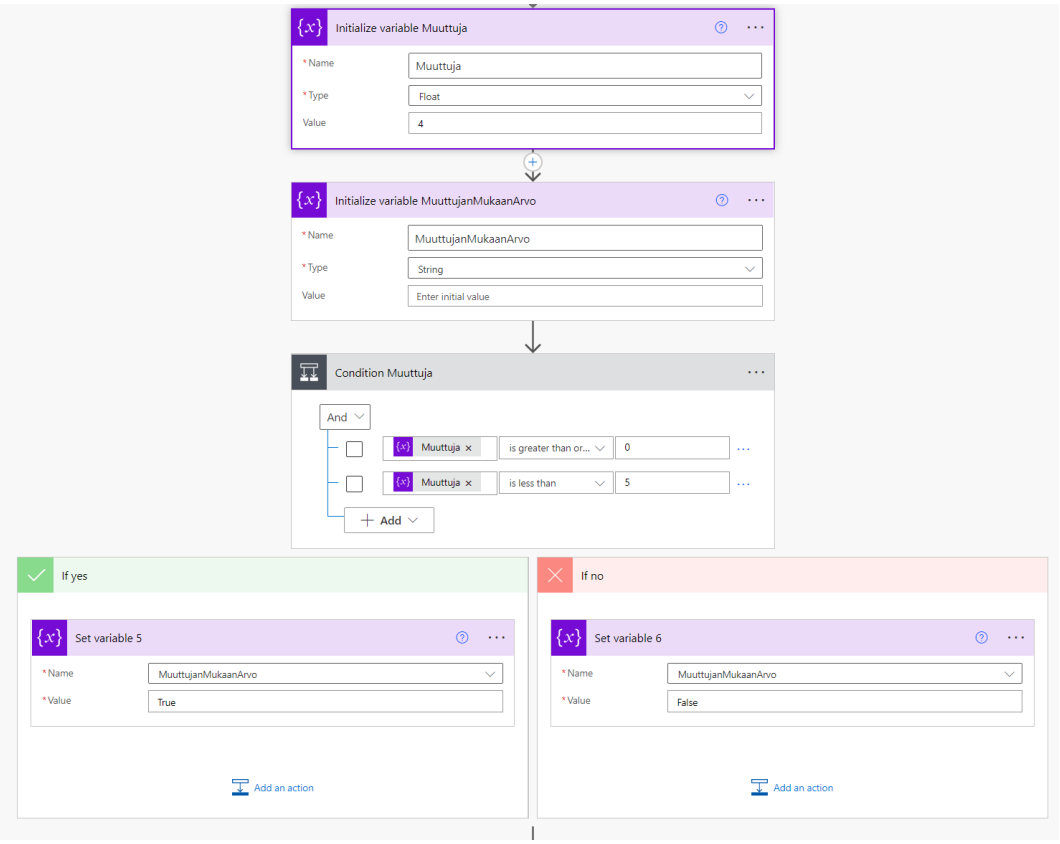
3.2.3 Ehdot

Ehtoja (Conditions) käytetään arvioimaan ja valitsemaan ne olosuhteet, joiden mukaan toimintoja suoritetaan. Ehtoja voidaan rajoittaa esimerkiksi ajan, käyttäjän syötteestä saatujen arvojen, luetun tiedoston tai muun lasketun arvon tuloksella. Ehdot voivat johtaa haarautumiin tai erilaisiin loogisiin päätöksiin, jotka toteutetaan riippuen saadusta tuloksesta. (Guilmette. 2020)

Arvojen tarkastukseen voidaan käyttää useita erilaisia lausekkeita. Näitä lausekkeita ovat esimerkiksi:

- And-lauseke vertaa kahta argumenttia ja antaa tulokseksi tosi, jos molemmat arvot ovat tosia
- Or-lauseke vertaa kahta argumenttia ja antaa tulokseksi tosi, jos kumpi tahansa arvo on tosi
- Equals-lauseke antaa tulokseksi tosi, jos kaksi arvoa ovat yhtä suuret
- Less-lauseke vertaa kahta argumenttia ja antaa tulokseksi tosi, jos ensimmäinen argumentti on pienempi kuin toinen

- Greater-lauseke vertaa kahta argumenttia ja antaa tulokseksi tosi, jos ensimmäinen argumentti on suurempi kuin toinen. (Microsoft 2021g.)



Kuva 8. Esimerkki ehto.

Kuva 8 sisältää kaksi muuttujaa, joiden nimet ovat Muuttuja ja MuuttujanMukaanArvo. Muuttuja sisältää arvon 4 ja MuuttujanMukaanArvo on tyhjä. Ehdossa tarkastellaan, jos Muuttuja on suurempi tai yhtä suuri kuin nolla mutta pienempi, kuin viisi asetetaan MuuttujanMukaanArvo True, muussa tapauksessa asetettu arvo on False.

3.3 CDS

CDS (Common Data Service) on tiedontallennusväline, joka mahdollistaa organisaatiota tallentamaan yritystoiminnan tietoja. Tiedot tallennetaan itsenäisenä kokonaisuutena, joka on sarja yhteen liittyviä merkintöjä ja kenttiä. (Guilmette 2020.)

Useat sovellukset voivat käyttää CDS:ään tallennettua tietoa, kuten esimerkiksi Dynamics 365, Power Apps, Power Automate ja Power BI. Edistyneet Power Automate ratkaisut vaativat pääsyn CDS:ään. (Guilmette 2020.)

3.4 Tietoyhdyskäytävä

Tietoyhdyskäytävä (Data Gateway) on tietokoneohjelmisto, joka asennetaan paikalliselle tietokoneelle. Tätä sovellusta käytetään mahdollistamaan yhteys Power Platform palvelujen ja paikallisten ympäristöjen tietolähteiden välillä. Tietoyhdyskäytävää voidaan käyttää esimerkiksi sallimaan Power Automatea lukemaan kohteita paikallisen SharePoint palvelimen luettelosta. (Guilmette 2020.)

Yhdyskäytäviä on kahta eri tyyppiä, jotka ovat:

- Paikallinen tietoyhdyskäytävä
- Paikallinen tietoyhdyskäytävä (henkilökohtainen tila)

Paikallisessa yhdyskäytävässä useat käyttäjät voivat muodostaa yhteyden paikallisiin tietolähteisiin. Tämä yhdyskäytävä sopii hyvin tilanteisiin, joissa useat käyttäjät tarvitsevat useita tietolähteitä. Henkilökohtaisessa tilassa vain yksi käyttäjä voi muodostaa yhteyden tietolähteisiin, eikä tietoyhdyskäytävää voi jakaa muiden kanssa. Tämä yhdyskäytävä sopii hyvin tilanteisiin, joissa on vain yksi raporttien luoja, eikä tietolähteitä tarvitse jakaa muiden käyttäjien kanssa. (Microsoft 2021h.)

3.5 Askeleet

Askeleilla (Steps) tarkoitettaan yksittäisiä evaluaatioita ja toimintoja, joita flow suorittaa. Askeleet järjestetään metodisella tavalla. Esimerkkejä flown askeleista ovat:

- Viestin lähetys Teams-kanavalle
- Sähköpostiviestin lukeminen ja lähettäminen
- Tiedoston tallennus ja uuden tiedoston luominen

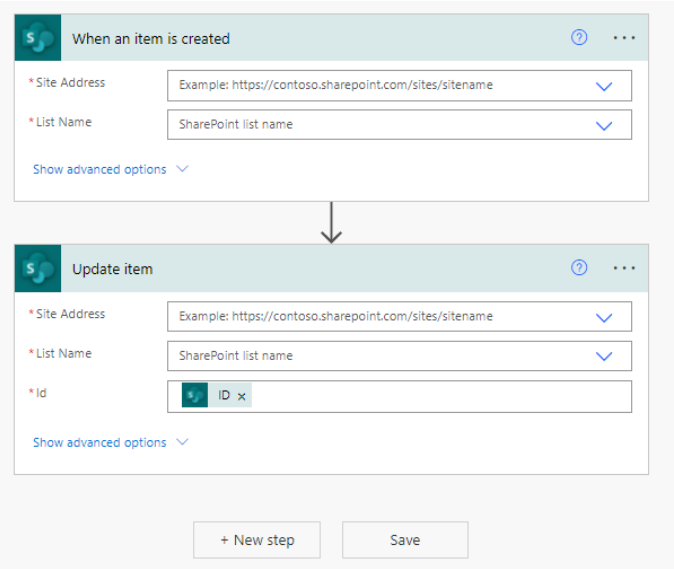
- Muuttujan tietyn arvon tarkastaminen

Flowt rakentuvat yhdestä tai useammasta askeleesta. (Guilmette 2020.)

3.6 Mallit

Mallit (Templates) sisältävät liittimiä, käynnistäjiä ja toimintoja, joiden tarkoituksena on ennalta määrätyn aikomuksen toteuttaminen (kuva 9). Mallit mahdollistavat standardoidun käyttöönoton ja yleisten komponenttien sekä konfiguroinnin uudelleenkäytön. Mallit mahdollistavat flown luomisen testiympäristössä, jonka jälkeen se voidaan siirtää ja ottaa käyttöön tuotantoympäristössä. Tällä tavoin voidaan pienentää väärin konfiguroidun flown riskiä. (Guilmette 2020.)

Power Automate sisältää mallikokoelman, josta on helppo valita skenaariota vastaava malli. Malleista voi luoda omanlaisensa työnkulun lisäämällä, muokkaamalla tai poistamalla käynnistimiä ja toimintoja. (Microsoft 2021a.)

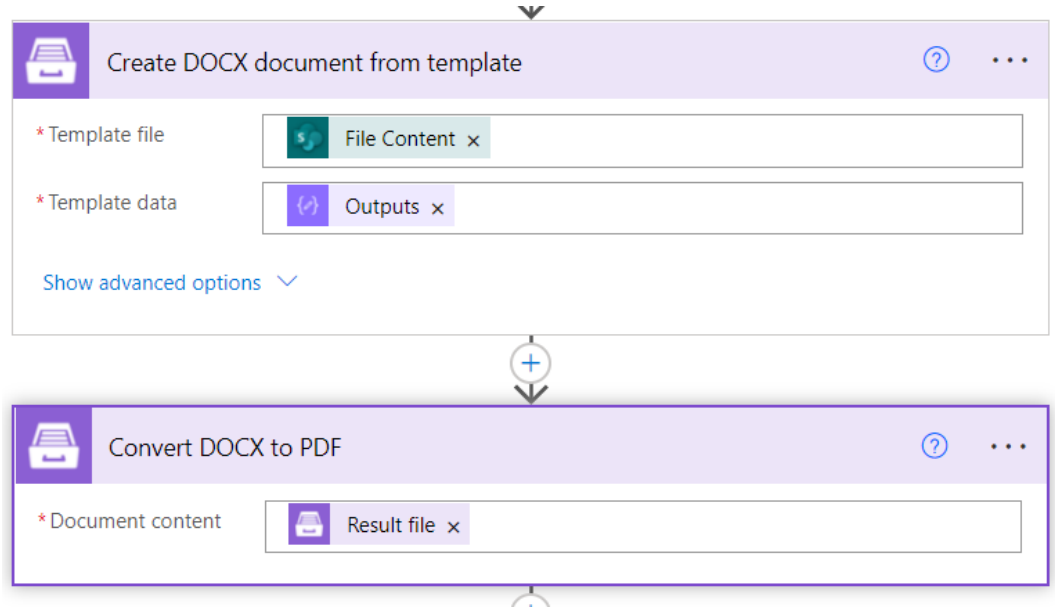


Kuva 9. Malli flow.

3.7 Plumsail

Plumsail on Power Automate ratkaisuja tarjoava yritys, joka on keskittynyt etenkin dokumenttien generointiin sekä web-lomakkeiden luomiseen. Dokumenttien generointi tapahtuu käyttäen Plumsail Documentsia ja web-lomakkeisiin käytetään Plumsail Formsia.

Plumsail Documents sisältää useita liittimiä, joiden avulla voidaan generoida Word, Excel, Power Point, HTML ja PDF-lomakkeita ja täyttää niitä SharePointista saaduilla tiedoilla. Sillä mahdollistetaan myös dokumenttien muokkaaminen riippuen haetusta tiedosta, esimerkiksi tyhjien kenttien piilotus ja numero- sekä valuuttakenttien muotoilu. (Plumsail 2022a.)



Kuva 10. Plumsailin liittimiä.

Kuvassa 10 on esimerkki kahdesta Plumsail-liittimestä. Ensimmäinen liitin täyttää Word-dokumentin (File Content) kootulla datalla (Outputs). Toinen liitin muuttaa luodun dokumentin PDF-muotoon.

Plumsail Forms mahdollistaa responsiivisten ja kustomoitavien lomakkeiden rakentamisen. Lomakkeet lisätään halutulle nettisivulle HTML-pienohjelmalla tai ne

voidaan jakaa linkillä. Lähetetyn lomakkeen tiedot voidaan hakea Power Automatella, josta ne voidaan lisätä esimerkiksi halutulle luettelolle. (Plumsail 2022b.)

4 ANGULARJS

AngularJS on Googlen ylläpitämä, vuonna 2012 julkaistu avoimen lähdekoodin JavaScript-ohjelmistokehys, jonka tarkoituksena on pääasiassa yksisivuisten web-sovellusten kehittäminen. AngularJS laajentaa HTML:ää antamalla sille malli-näkymä-kontrolleri (MVC) kapasiteetin, jonka avulla sivustojen kehitys ja testaus parantuvat. AngularJS mahdollistaa sen, että modernien ja monimutkaisten web-sovellusten kehittäminen olisi mahdollisimman helppoa ja kivutonta. AngularJS:n lähdekoodi on saatavilla ilmaiseksi GitHubista MIT-lisenssillä, mikä tarkoittaa, että kaikkien on mahdollista olla osallisena sen toiminnan edistämisessä.

AngularJS kirjasto toimii siten, että sillä luetaan HTML-elementteihin tehtyjen attribuuttien avulla, mitä toiminnallisuutta kyseiseen kohtaan halutaan, minkä jälkeen se yhdistää elementtiin toiminnallisuuden mukaisen syötteen muuttujien mukaisesti. Muuttujia voidaan merkitä joko staattisesti tai dynaamisesti. Staattisessa metodissa käytetään JavaScriptiä asettamaan arvo muuttujaan ja dynaamisessa metodissa arvot luetaan tietovarastosta käyttämällä esimerkiksi JSON:ia. (AngularJS 2021j.)

AngularJS:n kehittänyt tiimi on korostanut kirjaston koon pitämistä mahdollisimman pienenä, kun se on pakattu, jotta se ei aiheuta käyttäjälle hankaluuksia. Tämä toiminto tekee AngularJS:stä erityisen hyvän uusien prototyyppi ominaisuuksien luomisessa (Lerner 2013, 9).

4.1 MVC-arkkitehtuuri

MVC-arkkitehtuuri (Model-View-Controller) eli malli-näkymä-kontrolleri on ohjelmistoarkkitehtuurin periaate, jonka tarkoitus on jakaa sovellus kolmeen loogiseen osaan. Nämä kolme osaa ovat juuri edellä mainitut malli, näkymä ja kontrolleri. Jokainen näistä komponenteista on rakennettu käsittelemään tiettyä sovelluskehityksen näkökulmaa. MVC-arkkitehtuuri on yksi käytetyimmistä yleisen

standardin verkkokehityskehyksistä, joita käytetään rakentamaan skaalautuvia ja laajennettavia projekteja. (tutorialspoint 2021a.)

Mallikomponentti vastaa kaikesta dataan liittyvästä logiikasta, jonka kanssa käyttäjä on tekemisissä. Tällä voidaan tarkoittaa joko dataa, joka siirtyy näkymä- ja kontrollerikomponenttien välillä tai mitä tahansa muuta liiketoimintalogiikkaan liittyvää dataa. Esimerkkinä tästä voi olla asiakasobjekti, joka hakee asiakkaan tiedot tietokannasta, käsittelee niitä ja päivittää tiedot takaisin tietokantaan tai käyttää niitä renderöimään dataa. (tutorialspoint 2021a.)

Näkymäkomponenttia käytetään kaikkeen sovelluksen UI-logiikkaan. Esimerkiksi asiakasnäkymä pitää sisällään kaikki UI-komponentit, kuten tekstilaatikot, pudotusvalikot, valintaruudut ja painikkeet, joiden kanssa lopullinen käyttäjä on vuorovaikutuksessa. (tutorialspoint 2021a.)

Kontrollerit toimivat rajapintana malli- ja näkymäkomponenttien välillä. Niiden tarkoitus on prosessoida kaikkea liiketoimintalogiikkaa ja saapuvia pyyntöjä, käsitellä dataa käyttäen mallikomponenttia ja toimia vuorovaikutuksessa näkymien kanssa renderöidäkseen lopullisen tulosteen. Esimerkkinä asiakaskontrolleri käsittelee kaikki keskustelut ja syötöt, joita se saa asiakasnäkymältä ja päivittää ne tietokantaan käyttäen asiakasmallia. Samaa kontrolleria käytetään näyttämään asiakkaan dataa. (tutorialspoint 2021a.)

Jokainen osa on vastuussa vain yhdestä ja ainoasta asiasta. Malli vastaa dataa, näkymä UI:tä ja kontrolleri liiketoiminnanlogiikkaa. Paikan löytäminen uudelle koodille ja vanhan koodin löytäminen on helppoa tämän yksittäisen vastuun periaatteen avulla. Jokainen osa on niin riippumaton muista, kuin vain mahdollista. Tämä tekee koodista paljon enemmän moduuleista koostuvan, uudestaan käytettävän ja helposti ylläpidettävän (Seshadri & Green 2014, 3).

4.2 Angular-formly

Angular-formly on dynaaminen lomakekirjasto, joka mahdollistaa mukautettujen lomaketoimintojen rakentamisen (Kuva 11). Lomakekomponentit voidaan määritellä erikseen jokaisessa kentässä, sen sijaan, että määrittely tapahtuisi pelkästään HTML-pohjassa. (Formly 2022.)

Angular-formly tarjoaa useita etuja, kuten esimerkiksi lomakkeiden helpon muokkaamisen ja selkeämmän koodin. Se sisältää useita erilaisia lomaketoimintoja, kuten esimerkiksi kenttien kokojen muokkaus, kenttien piilotus, ikonien käyttö ja vahvistuksien lisäys. (Formly 2022.)

```
{//Singleline
"className": "col-md-12",
"key": "Title",
"type": "horizontal-Input",
    "templateOptions": {
        //"labelClass": "col-md-3 control-label",
        //"divClass": "col-md-9",
        "label": "Title",
        "placeholder": "Title",
        /*"addonLeft": {
          "class": "glyphicon glyphicon-tag"
          },
        "required": true*/
    },
},
```

Kuva 11. Title-kentän luominen Angular-formlyssa.

4.3 Direktiivit

Direktiivit (Directive) ovat DOM-elementin merkitsijöitä, kuten esimerkiksi attribuutti, elementin nimi tai kommentti, jotka kertovat HTML kääntäjälle minkälaista toiminnallisuutta kyseiseen DOM elementtiin halutaan lisätä. Direktiiveillä on myös mahdollista muuttaa DOM elementtiä ja sen lapsiobjekteja (children) täysin toisenlaisiksi. (AngularJS 2021e.)

Direktiivit ovat periaatteessa AngularJS:n tapa opettaa selaimelle ja HTML:lle uusia asioita, kuten vaikka klikkausten ja ehtojen ymmärtäminen tai uusien rakenteiden luominen ja suunnittelu (Seshadri & Green 2014, 9).

AngularJS sisältää useita sisäänrakennettuja direktiivejä. Direktiivit alkavat aina etuliitteellä ng- ja niitä ovat esimerkiksi:

- ng-app alustaa sovelluksen
- ng-controller asettaa näkymälle kontrollerin
- ng-click toteuttaa toiminnon elementtiä klikatessa
- ng-show / ng-hide näyttää / piilottaa direktiivin sisällön määritellyn arvon mukaan
- ng-if asettaa tai poistaa DOM elementit direktiivin sisällä määritellyn arvon mukaan. (AngularJS 2021e.)

Kuva 12. Esimerkki direktiivistä.

Kuvassa 12 on esimerkki direktiivistä. Tätä direktiiviä käytetään luettelon kautta sivupalkin määrittelyyn.

4.4 Datakytkentä

Datakytkennällä (Data-binding) tarkoitetaan automaattista datan synkronointia malli- ja näkymäkomponenttien välillä. AngularJS toteuttaa datakytkennän siten, että se mahdollistaa mallin käsittelemisen yhtenä aitona totuuden lähteenä sovellukselle. Näkymä on projektio mallista, joten kun malli muuttuu, näkymä kuvastaa muutosta. Tämä muutos toimii myös toiseen suuntaan eli jos näkymä muuttuu malli kuvastaa sen muutoksen. (AngularJS 2021b.)

Useimmat templaattijärjestelmät kytkevät dataa vain yhteen suuntaan, eli ne yhdistävät templaatti- ja mallikomponentteja yhteen näkymään. Yhdistämisen jälkeen muutokset malleissa eivät automaattisesti kuvastu näkymissä ja muutokset näkymissä eivät automaattisesti kuvastu malleissa. Tässä tilanteessa toiminnallisuuden takaamiseksi on kirjoitettava koodi, joka jatkuvasti synkronoi näkymää mallin kanssa ja mallia näkymän kanssa. (AngularJS 2021b.)

AngularJS:n templaatit toimivat eri tavalla useimpiin templaatteihin verrattuna. Ensimmäisenä templaatti käännetään selaimessa. Kääntämisprosessi saa aikaan livenäkymän. Kaikki muutokset näkymässä kuvastuvat välittömästi mallissa ja kaikki muutokset mallissa monistuvat näkymään. Tällä tavoin sovelluksen ohjelmointi yksinkertaistuu suuresti. Kontrolleri on täysin erossa ja tietämätön näkymästä. Tämä mahdollistaa vaivattoman testaamisen, koska kontrolleria on helppo testata, kun se on eristetty näkymästä ja siihen liittyvästä DOM/selain riippuvuudesta. (AngularJS 2021b.)

4.5 Riippuvuusinjektio

Riippuvuusinjektiolla (Dependency Injection) tarkoitetaan ohjelmistosuunnittelumallia, joka käsittelee sitä, miten komponentit pääsevät käsiksi riippuvuuksiinsa. Tällä riippuvuudella tarkoitetaan sovelluskomponenttia, jonka toiminnallisuuttaa tarvitsee jokin toinen sovelluskomponentti. (AngularJS 2021d.)

Pääasiassa on olemassa vain kolme tapaa, joilla objektit löytävät näiden riippuvuudet. Nämä kolme tapaa ovat:

1. Luominen sisäisesti riippuvuuteen
2. Etsiä tai osoittaa sitä globaalina muuttujana
3. Välittää se sinne missä sitä tarvitaan. (Lerner 2013, 149).

Riippuvuusinjektiolla käytetään kolmatta tapaa, sillä loput tavat voivat aiheuttaa erilaisia haasteita. Riippuvuusinjektio on malli, joka mahdollistaa kovakoodattu-

jen riippuvuuksien poistamisen näin tehden niiden poistamisesta ja muuttamisesta mahdollista ajon aikana (Lerner 2013, 149).

Riippuvuuksien muokkaus ajon aikana mahdollistaa eristäytyneiden alustojen luomisen, jotka ovat ideaalisia testaamiselle. Oikeita objekteja tuotantoalustalla voidaan vaihtaa testiobjekteihin testausalustalla. Funktionaalisesti malli injektoi tarvittavia riippuvaisia resursseja määränpäähän katsomalla automaattisesti riippuvuuden etukäteen ja toimittamalla määränpään riippuvuudelle (Lerner 2013, 149).

4.6 Moduulit

Moduulit (Module) ovat tapa, jolla AngularJS pakkaa kaiken oleellisen koodin yhden tietyn nimikkeen alle. AngularJS:n moduulissa on kaksi osaa. Ensimmäinen osa on moduulin kyky määritellä omat kontrollerit, palvelut, tehtaat ja direktiivit. Nämä ovat funktioita ja koodia, joihin pääse käsiksi moduulin kautta. Toinen osa on se, että moduuli voi olla myös riippuvainen toisista moduuleista, joka määritellään, kun moduuli ilmennetään. Tämä tarkoittaa, että AngularJS etsii moduulin tällä tietyllä nimikkeellä ja varmistaa, että kaikki mahdolliset moduulissa määritellyt funktiot, kontrollerit ja palvelut ovat saatavilla toisen moduulin kautta (Seshadri & Green 2014, 15).

Moduulien käyttäminen tarjoaa paljon etuja, kuten esimerkiksi globaalin nimitilan pitäminen puhtaana, testien helpottaminen, koodin jakaminen sovelluksien välillä ja koodin osien lataaminen halutussa järjestyksessä (Lerner 2014, 18).

4.7 Lausekkeet

Lausekkeet (Expression) ovat AngularJS:n tapa yhdistää dataa HTML-pohjaan. Lausekkeet voidaan kirjata aaltosulkeiden väliin tai direktiivien sisälle. AngularJS selvittää lausekkeen ja palauttaa arvon lausekkeen osoittamaan paikkaan. Lausekkeita voivat olla esimerkiksi {{ 1+2 }}, {{ expression }} tai direktiivin sisäinen ng-bind=”expression”. (W3 Schools 2021a.)

AngularJS:n ja JavaScriptin lausekkeet ovat hyvin samanlaisia, mutta niissä on silti selkeitä eroja, kuten esimerkiksi jos lausekkeet evaluoidaan scopen mukaan, määrittelemättömästä lausekkeesta ei aiheudu virheviestiä. AngularJS-lausekkeissa ei voida käyttää ehtoja, silmukoita tai poikkeuksia. (AngularJS 2021g.)

4.8 Kontrollerit

Kontrollerit (Controller) ovat vastuussa suurimmasta osasta UI-painotteisesta työstä. Kontrollereita voidaan yhdistää DOM:iin eri tavoilla, joista jokainen luo uuden kontrolleriobjektin rakentaja funktion mukaan. Yhdistäminen voidaan suorittaa ngController-direktiivillä, route-kontrollerilla ja käyttämällä tavallisen- tai komponenttidirektiivin kontrolleria. Kontrolleja käytetään esimerkiksi:

- Datan hakemiseen palvelimelta
- Tietyn datan näyttämiseen
- Esittelylogiikkaan, eli esimerkiksi mitä osia näytetään
- Käyttäjän ja sovelluksen väliseen vuorovaikutukseen.

(AngularJS 2021c.)

Kontrollereita käytetään lähes aina yhteydessä näkymän tai HTML:n kanssa. Ne ovat välikappaleita, jotka yhdistävät sovelluksen mallin ja näkymän yhdeksi kokonaisuudeksi (Seshadri & Green 2014, 17).

```
//console.clear();
console.debug('languageController.js -- START');
angular.module('MetronicApp')
    .controller('languageController', ['settings', '$rootScope', '$scope', 'settingsService', 'languageService', 'translationService', '$injector',
        function (settings, $rootScope, $scope, settingsService, languageService, translationService, $injector) {
            console.log('languageController START ');
            var ctrl = this;
            $scope.$watch(function () { return $rootScope.settings.languages; }, function (languages) {
            //console.log('scope watch - languages: ', languages);
            ctrl.languages = languages;
            }, true);
            $scope.$watch(function () { return $rootScope.settings.activeLanguage; }, function (language) {
            console.log('languageController - watch activeLanguage: ', language);
            //ctrl.activeLanguage = languageService.getActiveLanguage().then(function (language) {
              ctrl.activeLanguage = language;
        });

            var tmhDynamicLocale = null;
            if (typeof modules != 'undefined' && _.contains(modules, 'tmh.dynamicLocale')) {
                tmhDynamicLocale = $injector.get('tmhDynamicLocale')
            }

        ctrl.setActiveLanguage = setActiveLanguage;
        function setActiveLanguage(lang) {
            // console.log('setActiveLanguage - lang: ', lang);
            // languageService.setActiveLanguage(lang);

            // ctrl.activeLanguage = lang;

            settings.activeLanguage = lang;

                moment.locale(lang.locale);

                if (tmhDynamicLocale) {
                    tmhDynamicLocale.set(lang.locale);
                    setDatetimePickerLocale(lang.locale);
                }

            translationService.getCommonTranslationsForLanguage().then(function (data) {
                console.log('translationService getCommonTranslationsForLanguage data: ', data);
                //console.log('data.then');
                if (data.value) {
                    settings.commonTranslations = data.value;
                        console.log('settings.commonTranslations: ', settings.commonTranslations);

                        broadcast($rootScope, 'setActiveLanguage');
                }
                if (data.data) {
                    settings.commonTranslations = data.data;
                        console.log('settings.commonTranslations: ', settings.commonTranslations);

                        broadcast($rootScope, 'setActiveLanguage');
                }
                if (data.then) {
                    data.then(function (data2) {
                        console.log('translationService - getCommonTranslationsForLanguage data2: ', data2);
                        settings.commonTranslations = data2.data;

                            broadcast($rootScope, 'setActiveLanguage');
                    })
                }
            });
        }
    }])
```

Kuva 13. Esimerkki kontrollerista ja injektiosta.

Kuvassa 13 on esimerkki kontrollerin ja injektion toiminnasta. Tällä koodilla suoritetaan kielikäännösten määrittely ja hakeminen sivustolle.

4.9 Scope

AngularJS:ssä scope on sisäänrakennettu objekti, joka sisältää sovellusdataa ja metodeja. Kontrollerifunktion sisäiseen scope objektiin on mahdollista luoda ominaisuuksia, joihin määrätään arvo tai funktio. Scopet toimivat hiearkisena rakenteena samalla tyylillä kuin sovelluksen DOM-rakenne. (AngularJS 2021n.) Scope toimii liimana kontrollerin ja näkymän välillä. Sillä siirretään dataa kontrollerista näkymään ja toisinpäin (Lerner 2014, 20).

Scopet on mahdollista luoda sisäkkäisesti rajaamaan pääsyä sovelluskomponenttien ominaisuuksiin samalla säilyttämällä pääsyn jaettuihin malliominaisuuksiin.

Sisäkkäiset scopet voivat olla, joko alemman tason scopeja tai eristettyjä scopeja. Eristetyt scopet eivät peri ominaisuuksia, toisinkuin alemman tason scopet, jotka perivät ominaisuuksia ylemmän tason scopeista. (AngularJS 2021n.)

AngularJS sisältää aina vain yhden root scopen mutta alemman tason scopeja voi olla useita. Direktiiveillä on mahdollista luoda uusia alemman tason scopeja. Uudet scopet lisätään ylemmän tason scopen alle, josta syntyy rinnakkainen rakenne DOM:in vierelle. (AngularJS 2021n.)

Kuva 14. Esimerkki scope.

Kuva 14 havainnollistaa scopen käyttöä Angular-formlyn lomakekentässä. Kuvassa yhdistetään kentät Sukunimi ja Etunimi yhdeksi kokonimeksi Title-kenttään.

4.10 Vahvistaminen

Vahvistamisella (Validation) tarkoitetaan ilmoitusta väärästä syötteestä ennen lomakkeen lähetystä. Käyttäjä saa välittömästi palautteen, jonka avulla käyttäjän itse on helppo ja nopea korjata tekemänsä virhe. Palvelinpuolen vahvistus ei mahdollista välitöntä palautetta, minkä takia etenkin käyttäjäkokemuksen kannalta AngularJS:n tarjoamat vahvistukset ovat tarpeellisia. Palvelinpuolen vahvis-

tus on silti erittäin tärkeä osa käyttäjäkokemusta ja turvallista sovellusta. (AngularJS 2021i.)

Kaikki syöttökentät voivat käyttää, joitakin perusvahvistuksista. Esimerkkejä vahvistusvaihtoehdoista ovat:

- Required-kenttä on pakollinen ja se on täytettävä ennen kuin tallennus tai lähetys on mahdollista (kuva 15).
- ng-minlength kentälle on syötettävä vähintään tietty määrä merkkejä
- ng-maxlength kentälle voi syöttää vain tietyn määrän merkkejä
- type=”number”-kenttä voi sisältää vain numeroita
- type=”url”-kenttä voi sisältää vain URL-osoitteen

AngularJS:ssä on myös mahdollista luoda omia vahvistuksia käyttämällä direktiivejä laukaisemaan esimerkiksi Ajax-pyynnön (Lerner 2014, 46–47).

```
{//Singleline
"className": "col-md-12",
"key": "Title",
"type": "horizontal-Input",
    "templateOptions": {
        //"labelClass": "col-md-3 control-label",
        //"divClass": "col-md-9",
        "label": "",
        "placeholder": "",
        "addonLeft": {
          "class": "glyphicon glyphicon-tag"
          },
        "required": true
    },
},
```

Kuva 15. Esimerkki vahvistamisesta Angular-formlyn lomakekentässä.

4.11 Suodattimet

Suodattimia (Filter) käytetään muotoilemaan dataa, jota käyttäjälle näytetään. AngularJS sisältää useita sisäänrakennettuja suodattimia mutta myös omien suodattimien määrittely on mahdollista (Lerner 2014, 37). Suodattimia on mahdollis-

ta käyttää lausekkeissa, näkymissä, kontrollereissa ja palveluissa (AngularJS 2021h).

Suodattimien kutsuntaan käytetään pystyviivaa, joka sijoitetaan aaltosulkeiden väliin (Lerner 2014, 37). Suodattimia voidaan käyttää kolmella eri tavalla, jotka ovat:

1. Yksinkertainen lausekkeen suodatin {{ expression | filter }}
2. Ketjutettu suodatin {{ expression | filter1 | filter2… }}
3. Suodatin argumentilla {{ expression | filter:argument1:argument2… }}

(AngularJS 2021g.)

Suodattimet ovat erittäin hyödyllisiä päivämäärien ja valuuttojen muotoilussa paikallisiin arvoihin. Suodattimia voidaan myös hyödyntää esimerkiksi luetteloiden järjestyksen muotoiluun ja datan esille tuontiin. (W3 Schools 2021b.)

```
'filter':
[
    {
        'field': 'HenkiloId',
        'idfield': 'HenkiloId',
        'criteria': 'eq',
        'value': '', // from url
        //'type': 'or',

    },
],
```

Kuva 16. Esimerkki suodatin.

Kuva 16 havainnollistaa suodattimen käyttöä Angular-formlyn lomakekentässä. Kuvassa suodatetaan hakusarake HenkiloId:n mukaan luettelolla näkyvät tiedot.

4.12 Palvelut

Palvelut (Service) ovat korvattavia objekteja, jotka yhdistetään riippuvuusinjektiolla. Niitä voidaan käyttää direktiiveissä, kontrollereissa, suodattimissa ja toisissa

palveluissa. Palveluiden pääfunktio on koodin organisointi ja jakaminen sovelluksessa. (AngularJS 2021l.)

AngularJS sisältää useita sisäänrakennettuja palveluita, mutta omien palveluiden luominen on myös mahdollista. Omia palveluita tarvitaan monimutkaisten sovellusten rakentamiseen. Oman palvelun luominen vaatii sen rekisteröintiä, jonka jälkeen siihen voidaan viitata ja se voidaan ladata riippuvuutena. (Lerner 2014, 157.)

AngularJS:ssä palveluiden luomiseen on viisi eri tapaa, jotka ovat:

- Constant()
- Value()
- Service()
- Factory()
- Provider()

(Lerner 2014, 163–164).

Constant on arvo, joka voidaan injektoida minne tahansa ja sen arvo voidaan muuttaa vain ohjelmallisesti. Value on yksinkertainen injektoitava arvo, joka voi olla merkkijono, numero tai funktio. Service on yksinkertainen injektoitava rakentaja, jonka AngularJS luo vain kerran. Factory on injektoitava funktio, joka muistuttaa palvelua. Provider mahdollistaa monimutkaisia luonti, funktio ja konfigurointia vaihtoehtoja. Varsinaisesti provider on konfiguroitava factory. (AngularJS 2021k.)

```
angular.module('MetronicApp')
    .service('loginService_o365', ['GraphService', '$http', 'settings',
        function(GraphService, $http, settings) {


            let vm = this;

            // View model properties
            vm.displayName;
            vm.requestSuccess;
            vm.requestFinished;

            // View model methods
            vm.login = login;
            vm.logout = logout;
            vm.isAuthenticated = isAuthenticated;
            vm.initAuth = initAuth;


            ////////////////////////////////////
            // End of exposed properties and methods.

            function initAuth() {
                // Check initial connection status.
                if (localStorage.auth) {
                    processAuth();
                } else {
                    let auth = hello('aad').getAuthResponse();
                    if (auth !== null) {
                        localStorage.auth = angular.toJson(auth);
                        processAuth();
                    }
                }
            }

            // Auth info is saved in localStorage by now, so set the default headers and user properties.
            function processAuth() {
                let auth = angular.fromJson(localStorage.auth);

                // Check token expiry. If the token is valid for another 5 minutes, we'll use it.
                let expiration = new Date();
                expiration.setTime((auth.expires - 300) * 1000);
                if (expiration > new Date()) {

                    // Add the required Authorization header with bearer token.
                    $http.defaults.headers.common.Authorization = 'Bearer ' + auth.access_token;

                    // This header has been added to identify our sample in the Microsoft Graph service. If extracting this code for your project please remove.
                    //$http.defaults.headers.common.SampleID = 'angular-connect-rest-sample';

                    if (localStorage.getItem('user') === null) {

                        // Get the profile of the current user.
                        GraphService.me().then(function(response) {

                            // Save the user to localStorage.
                            let user = response.data;
                            localStorage.setItem('user', angular.toJson(user));

                            settings.userName = user.displayName;
                            vm.displayName = user.displayName;
                            vm.emailAddress = user.mail || user.userPrincipalName;
                        });
                    } else {
                        let user = angular.fromJson(localStorage.user);

                        settings.userName = user.displayName;
                        vm.displayName = user.displayName;
                        vm.emailAddress = user.mail || user.userPrincipalName;
                    }
                }
            }

            vm.initAuth();

            function isAuthenticated() {
                return localStorage.getItem('user') !== null;
            }

            function login() {
                GraphService.login();
            }

            function logout() {
                console.debug('logout');
                GraphService.logout();
                console.debug('localStorage: ', localStorage);
            }
        }]);
```

Kuva 17. Esimerkki palvelusta.

Kuva 17 sisältää esimerkin palvelun käytöstä. Tällä palvelulla suoritetaan kirjautuminen sivustolle.

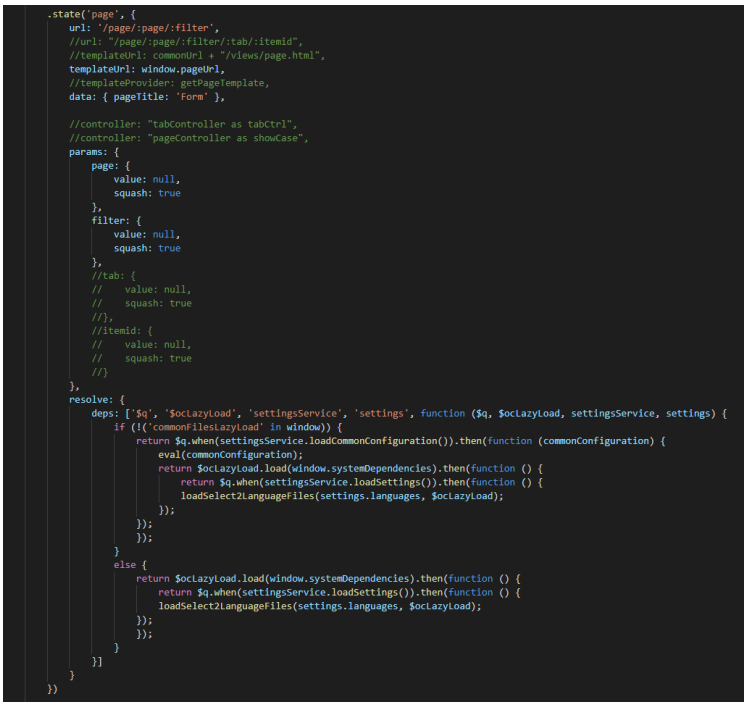
4.13 Reititys

AngularJS:ssä reititystä (Routing) käytetään yksisivuisten sovellusten navigointiin, kun halutaan siirtyä yhdeltä sivulta toiselle. Reitityksellä mahdollistetaan sovelluksen pitäminen yksisivuisena luomalla erilaisia URL-osoitteita eri sisällölle sovelluksessa. Reitityksessä käytettään ngRoute-moduulia erilaisten sivujen hakemiseen ilman, että koko sovellusta tarvitsee ladata uudelleen. (AngularJS 2021o.)

ngRoute on eritelty AngularJs:n ohjelmistokehyksestä, joten se käyttö vaatii ngRoute-moduulin lataamista sekä siihen viittaamista sovelluksessa (Lerner 2014, 136). ngRoute sisältää neljä erilaista moduulikomponenttia, jotka ovat:

1. ngView, direktiivi, jota käytetään täydentämään $route-palvelua sisällyttämällä siihen reitin templaatti
2. $routeProvider, käytetään reittien konfiguroinnissa
3. $route, käytetään yhdistämään URL-osoitteita kontrollereihin ja näkymiin
4. $routeParams, mahdollistaa reittiparametrien palauttamisen.

(AngularJS 2021a.)



```
.state('page', {
    url: '/page/:page/:filter',
    //url: "/page/:page/:filter/:tab/:itemid",
    //templateUrl: commonUrl + "/views/page.html",
    templateUrl: window.pageUrl,
    //templateProvider: getPageTemplate,
    data: { pageTitle: 'Form' },

    //controller: "tabController as tabCtrl",
    //controller: "pageController as showCase",
    params: {
        page: {
            value: null,
            squash: true
        },
        filter: {
            value: null,
            squash: true
        },
        //tab: {
        //    value: null,
        //    squash: true
        //},
        //itemid: {
        //    value: null,
        //    squash: true
        //}
    },
    resolve: {
        deps: ['$q', '$ocLazyLoad', 'settingsService', 'settings', function ($q, $ocLazyLoad, settingsService, settings) {
            if (!('commonFilesLazyLoad' in window)) {
                return $q.when(settingsService.loadCommonConfiguration()).then(function (commonConfiguration) {
                    eval(commonConfiguration);
                    return $ocLazyLoad.load(window.systemDependencies).then(function () {
                        return $q.when(settingsService.loadSettings()).then(function () {
                            loadSelect2LanguageFiles(settings.languages, $ocLazyLoad);
                        });
                    });
                    });
                }
                else {
                    return $ocLazyLoad.load(window.systemDependencies).then(function () {
                        return $q.when(settingsService.loadSettings()).then(function () {
                            loadSelect2LanguageFiles(settings.languages, $ocLazyLoad);
                        });
                        });
                    }
                }]
            }
        })
```

Kuva 18. Esimerkki reitityksestä.

Kuva 18 sisältää esimerkin reitityksestä. Tällä koodilla suoritetaan lomakkeiden pohjien määritys.

4.14 Testaaminen

AngularJS on rakennettu siten, että koko AngularJS:n ohjelmistokehystä on mahdollisimman helppo testata. Testaaminen tapahtuu yleensä hyödyntämällä Karma-sovellusta tai Jasmine-viitekehystä. AngularJS:ssä voidaan suorittaa yksikkötestauksia erikseen kontrollereilla ja direktiiveillä. (AngularJS 2021m). AngularJS:ssä on myös mahdollista tehdä E2E-testauksia (end of end testing), jolla tarkoitetaan testaamista käyttäjän näkökulmasta (AngularJS 2021f).

AngularJS sisältää ngMock-moduulin, jolla on mahdollista suorittaa harjoitustestejä. ngMock jäljittelee AngularJS-palveluita yksikkötestin sisällä ja synkronoituu muiden moduulien kanssa. ngMock-moduuli pystyy myös jäljittelemään XHR-pyyntöjä käyttämällä $httpBackend:iä. (AngularJS 2021m.)

AngularJS:ssä E2E-testaamiseen on rakennettu oma Node.js ohjelma nimeltään Protractor. Protractor käyttää Jasminea testisyntakseihin. Yksikkö- ja E2E-testauksissa luodaan testitiedosto, joka sisältää lohkoja, jotka kertovat sovelluksen vaatimukset. Lohkot koostuvat käskyistä ja odotuksista. Käskyjen tarkoitus on kertoa Protractorille mitä sen pitää tehdä sovellukselle ja odotusten tarkoitus on vakuuttaa jotakin sovelluksen tilasta. Odotuksen epäonnistuessa ajo merkitsee epäonnistuneen lohkon ja jatkaa seuraavan lohkoon. (AngularJS 2021f.)

5 PROJEKTIN TOTEUTUS

Tässä luvussa käydään läpi mitä vaaditaan projektin rakentamiseen ja miten kaikki edellä mainitut osat saadaan koottua yhtenäiseksi kokonaisuudeksi. Itse rakennettua projektia ei käytetä tässä osiossa havainnollistamaan työn kulkua, sillä se on jo asiakkaan käytössä. Työnkulkua havainnollistetaan erilaisilla esimerkeillä ja kuvilla, jotka ovat peräisin esimerkkiprojektista, jonka tarkoitus on tuoda esille järjestelmän toimintaa ja konfigurointia.

5.1 Sopivin 365

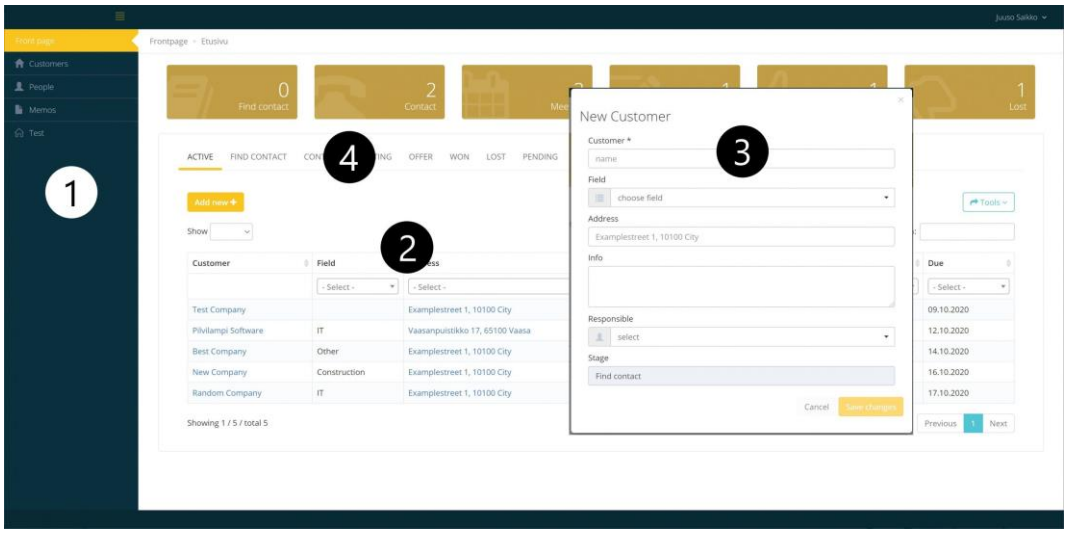
Sopivin 365 on Pilvilammen oma sovellusalusta, jossa kaikki yrityksen tiedot ja dokumentit tallennetaan SharePointin luetteloihin ja tiedostokirjastoihin. Kaikki tiedot pysyvät näin yrityksen omassa ympäristössä, eivätkä kärsi mahdollisista teknisistä ongelmista Pilvilammen puolella. Järjestelmään päästään käsiksi omalla O365-tunnuksella, joka mahdollistaa kertakirjautumisen (Single sign-on, SSO).

Sopivin 365 käyttöliittymä on suunniteltu hallinnoimaan suuri määriä dataa ja korvaamaan SharePointin oletuskäyttöliittymä. SharePointin oletus käyttöliittymällä ei ole mahdollista yhdistää dataa luomaan yhtenäisiä ja laajoja näkymiä tarpeellisista tiedoista. Sopivin 365 ei sulje pois mitään SharePointin sisäisiä ominaisuuksia, vaan mahdollistaa uusien ominaisuuksien käytön.

Konfigurointiin vaadittavat tiedostot ladataan Amazon S3 CDN tietoverkosta, joka on paremmin optimoitu, kuin tiedostojen lataaminen suoraan tiedostokirjastosta. Tämä on Microsoftin suosittelema menettelytapa. Järjestelmän konfigurointi tapahtuu käyttämällä AngularJS-kirjastoa, sekä BootStrap HTML-pohjaa. Tietojen hakemiseen ja säilyttämiseen sivustolla käytetään O365 REST-rajapintaa ja liike-elämän logiikan rakentamiseen käytetään O365 Flowta. Integraatiot tapahtuvat, joko käyttäen Flowta tai Layer2 CloudConnector-sovellusta.

5.2 Sopivin 365 käyttöliittymä

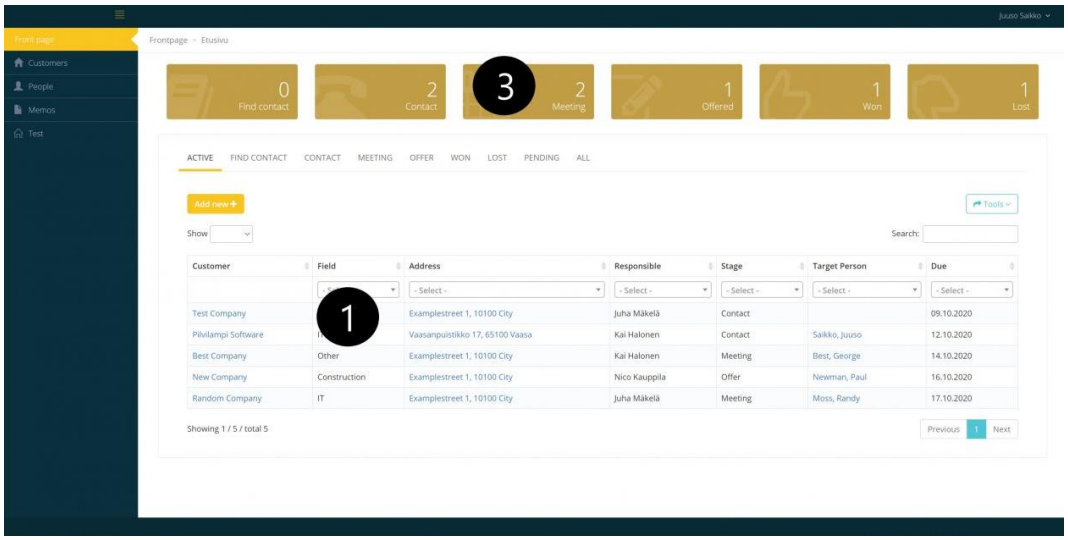
Sopivin 365 käyttöliittymä koostuu neljästä osasta, jotka ovat navigaatiovalikko, sivu, ponnahdusikkunasivu ja välilehti. Navigaatiovalikko sijaitsee sivun vasemmassa reunassa ja mahdollistaa helpon navigoinnin eri sivujen välillä. Navigaatiovalikko sisältää kaikki tärkeimmät sivut, kuten esimerkiksi asiakkaat, myynti tai projektinhallinta. Valikko sisältää myös etusivun, jolla voidaan tuoda esille yritykselle kriittistä informaatiota.



Kuva 19. Sopivin 365 käyttöliittymän etu- ja ponnahdusikkunasivu.

Kuva 19 esittää miltä käyttöliittymän etusivu voisi näyttää. Kuvassa on nähtävillä navigointivalikko (1), sivu (2), ponnahdusikkunasivu (3) ja välilehdet (4).

Sivulla tarkoitetaan navigointivalikosta tai hyperlinkistä avautuvaa sivua. Sivut jaetaan lohkoihin, joiden kokoa ja tyyliä voi konfiguroida. Jokaisessa lohkossa on mahdollista näyttää dataa käyttämällä luetteloita, lomakkeita tai ”dashboard”-ominaisuutta. ”Dashboard”-ominaisuutta käytetään suurten kokonaisuuksien esille tuontiin, esimerkiksi näyttämällä kaikkien luettelokohteiden lukumäärä. Luettelot näyttävät useiden kohteiden tiedot ja lomakkeilla näytetään ja editoidaan yhden tietyn kohteen tietoja. Sivulle on myös mahdollista lisätä tietolaatikoita, joita käytetään järjestelmän toiminnan selkeyttämiseen.

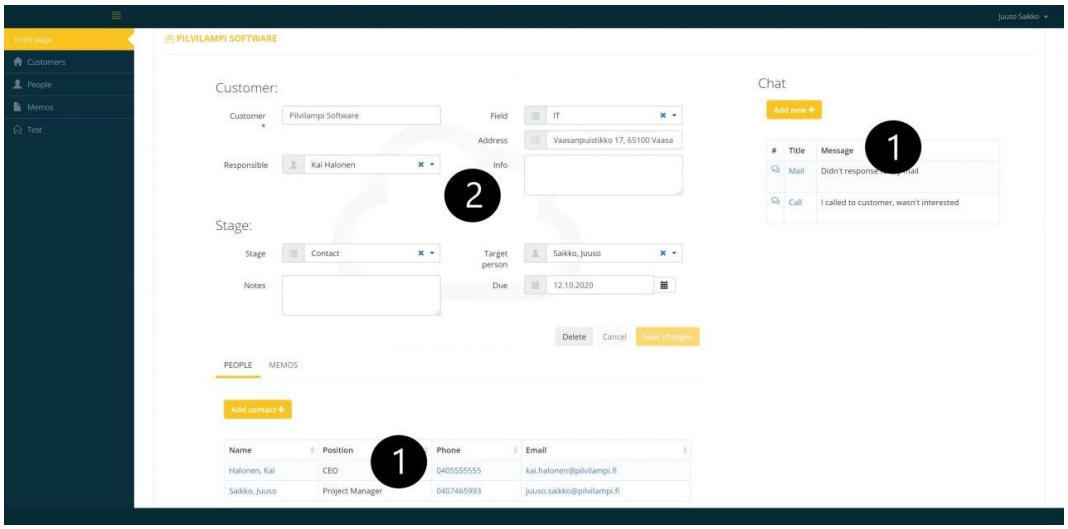


Kuva 20. Sopivin 365 käyttöliittymän etusivu.

Kuva 20 esittää miltä käyttöliittymän etusivu voisi näyttää. Kuvassa on nähtävillä luettelo (1) ja ”dashboard”-ominaisuus (3).

Ponnahdusikkunasivua käytetään pääosin uuden tiedon lisäämiseen luettelolle mutta myös jo luettelolla sijaitsevan tiedon editointiin. Uuden tiedon lisääminen luettelolle tapahtuu luettelon ”Lisää uusi” napista, jonka nimeä on mahdollista konfiguroida. Luettelokohteiden muokkausta varten toimiva ponnahdusikkunasivu avataan, joko luettelolla sijaitsevasta ikonista tai hyperlinkistä.

Välilehtiä käytetään suodatamaan erilaisia tarpeita omiksi luetteloiksi ja lomakkeiksi. Välilehdet pitävät järjestelmän navigoinnin sujuvana, sillä kaikkia luetteloita ja lomakkeita ei tällöin tarvitse saada mahtumaan yhdelle sivulle. Esimerkiksi Tarjous-luettelon kanssa voidaan haluta tuoda esille aktiiviset, voitetut ja hävityt tarjoukset. Tällöin on järkevämpää luoda jokaiselle luettelolle oma suodatettu välilehti sen sijaan, että kaikki kolme löytyisivät samalta välilehdeltä.



Kuva 21. Sopivin 365 käyttöliittymän asiakassivu.

Kuva 21 esittää miltä käyttöliittymän asiakassivu voisi näyttää. Kuvassa on nähtävillä luettelot (1) ja lomake (2).

5.3 Projektin konfigurointi: SharePoint

Sopivin 365 käyttöliittymän konfigurointi aloitetaan SharePointin asetuksista, koska kaikki tiedot tallennetaan SharePointin luetteloihin ja dokumenttikirjastoihin. SharePoint toimii pohjana projektin rakentamiselle, ja sen ympärille on helppo konfiguroida muita osia.

Jotta muiden osien konfigurointi onnistuisi, täytyy SharePointiin luoda sarakkeita, sisältötyyppejä, luetteloita ja dokumenttikirjastoja. SharePoint sivusto sisältää automaattisesti useita olemassa olevia sarakkeita, joita voidaan käyttää sivuston konfigurointiin mutta omien sarakkeiden luomista vaaditaan monimutkaisten järjestelmien rakentamiseen. Sarakkeet on hyvä nimetä järkevästi, jotta niiden käyttö muissa konfigurointivaiheissa olisi mahdollisimman suoraviivaista. Saraketyyppi valitaan kyseisen sarakkeen tarpeiden mukaan. Esimerkiksi, varastoitava data sisältää pelkästään lyhyen määrän tekstiä, tällöin käytetään yhden tekstirivin saraketta (Single line of text), jos taas varastoitava data sisältää pelkkiä numeroita tai matemaattiset laskelmat ovat tarpeellisia käytetään numerosaraketta (Number).

Sarakkeita voidaan luonnin jälkeen lisätä omille luetteloillensa, mutta jos useat luettelot käyttävät samoja sarakkeita on parempi luoda sisältötyyppi luettelolle. Tämä mahdollistaa helpomman konfiguroinnin ja muutosten tekemisen. Sisältötyyppi nimetään ja sen asetukset muutetaan vastaamaan sisältötyypin tarpeita. Sisältötyypille voidaan lisätä kaikki tarvittavat sarakkeet, josta ne ovat helposti löydettävissä.

Sivustolle täytyy luoda luetteloita ja dokumenttikirjastoja, jotta Sopivin 365:n konfigurointi olisi mahdollisimman tehokasta. Luetteloita käytetään pääosin tiedon säilytykseen, kun taas dokumenttikirjastoja tarvitaan dokumenttien säilytykseen. Luettelot ja kirjastot on helpointa nimetä niille tarkoitettujen sisältötyyppien mukaan. Asetuksien muokkausten jälkeen luettelolle tai kirjastolle voidaan lisätä sitä vastaava sisältötyyppi.

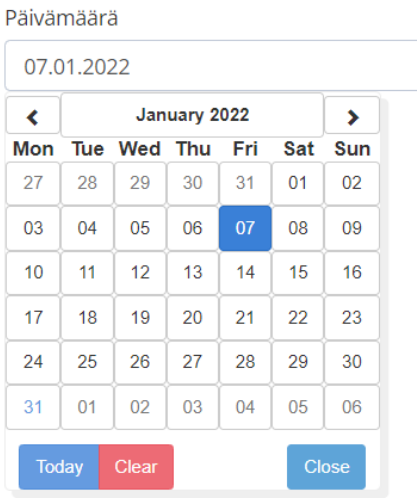
5.4 Projektin konfigurointi: AngularJS

Kun projektin SharePoint-osa on saatu valmiiksi, voidaan aloittaa ohjelmointivaihe. Sopivin 365 käyttöliittymän pohja koostuu erilaisista BootStrap-malleista, joita voidaan muokata haluttuun muotoon. Muokkaamalla malleja saadaan aikaiseksi erilaisia pohjia luetteloille ja lomakkeille. Esimerkiksi, kuinka monta ”dashboard”-laatikkoa halutaan tuoda esille vierekkäin tai kuinka monta nappia luettelolla tarvitaan. Luetteloiden ja lomakkeiden lopulliseen muotoiluun käytetään tarvittaessa CSS:ää.

Ennen kuin sivustolle voidaan lisätä luetteloita ja lomakkeita, täytyy sinne luoda sivuja. Sivut jaetaan lohkoihin, joiden kokoa ja tyyliä voidaan muokata. Lohkoja käytetään tuomaan esille haluttuja ”dashboard”-laatikoita, luetteloita ja lomakkeita. Sivun konfiguroinnissa ”dashboard”-laatikoille, luetteloille ja lomakkeille asetetaan omat sijainnit sivulla, jonka jälkeen niitä voidaan lähteä rakentamaan.

Luetteloiden ja lomakkeiden lisäämiseen käytetään AngularJS:än palikoita, joita muokkaamalla saadaan rakennettua yhtenäinen käyttöliittymä. Tärkein näistä

palikoista on Angular-formly, joka mahdollistaa mukautettujen lomaketoimintojen rakentamisen. SharePointiin lisättyjä luetteloita ja sarakkeita käytetään lähteinä luotaville osille. Esimerkiksi, lomakkeelle tai luettelolle halutaan päivämääräkenttä, tällöin SharePointiin luodaan päivämääräsarake, joka haetaan koodissa ja asetetaan halutulle lomakkeelle tai luettelolle (kuva 22).



Kuva 22. Päivämääräkenttä lomakkeella.

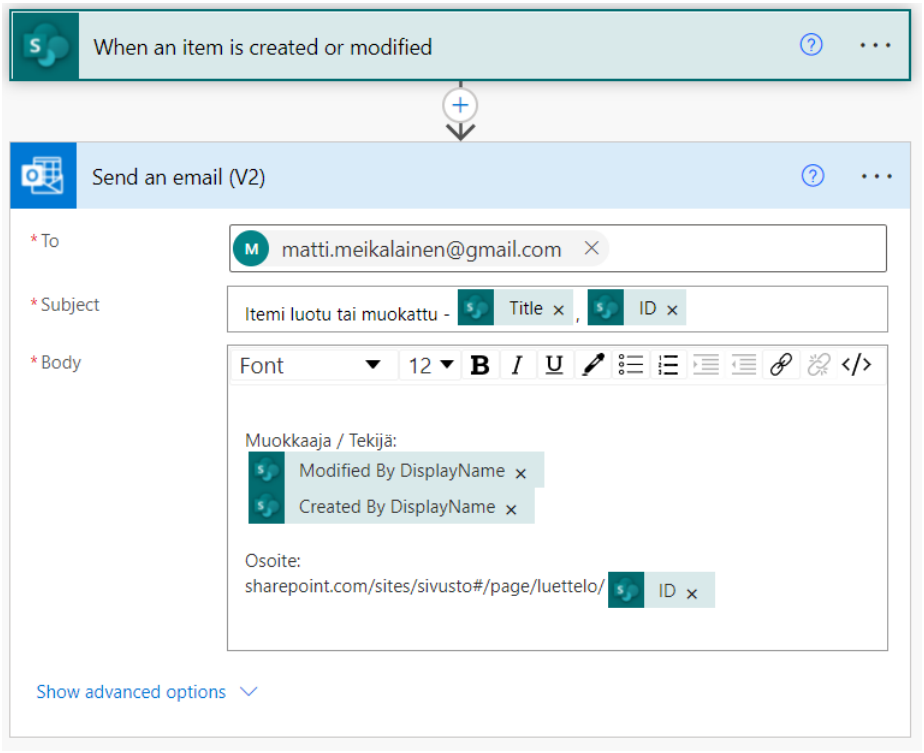
5.5 Projektin konfigurointi: Power Automate

Järjestelmää ohjelmoitaessa saatetaan törmätä ominaisuuksiin, joita ei ole mahdollista rakentaa käyttämällä AngularJS:sää. Yleensä nämä ominaisuudet vaativat SharePointin ulkopuolista automaatiota, johon käytetään Power Automate sovellusta. Power Automatea käytetään Flown rakentamisen, joka mahdollistaa sivuston toimintojen ja työnkulun automatisoinnin. On myös mahdollista, että järjestelmä ei vaadi minkäänlaista ulkopuolista automaatiota, vaan järjestelmän rakentaminen onnistuu pelkästään edellä mainituilla SharePoint ja AngularJS konfiguraatioilla.

Flown rakentaminen aloitetaan käynnistäjästä. Käynnistäjä valitaan sen perusteella mitä Flown halutaan tekevän. Esimerkiksi, halutaan, että Flow käynnistyy aina, kuin luettelolle luodaan uusi kohde tai kun vanhaa kohdetta muokataan, tällöin paras tapa on käyttää ”When an item is created or modified” käynnistä-

jää. Käynnistäjä yhdistetään halutulle sivustolle ja luettelolle, jonka jälkeen uudet kohteet ja muutokset SharePoint-luettelolla käynnistävät Flown.

Kun Flown käynnistäjä on toiminnassa, voidaan alkaa yhdistämään liittimiä, toimintoja ja ehtoja. Niitä yhdistetään, kunnes päästään haluttuun lopputulokseen, joka voi olla esimerkiksi sähköpostiviestin lähetys tietylle henkilölle tai halutun luettelon päivittäminen.



Kuva 23. Esimerkki flow.

Kuvassa 22 havainnollistetaan miltä flow voisi esimerkiksi näyttää. Käynnistäjänä toimii kohteen lisäys tai muokkaus tietyllä listalla. Tämän jälkeen lähetetään sähköpostiviesti, jossa ilmoitetaan kohteen Title, ID, muokkaaja ja luoja. Viestille lisätään myös kohteen osoite, josta päästään helposti käsiksi kohteeseen.

6 YHTEENVETO

Opinnäytetyön tavoitteena oli CRM-järjestelmän rakentaminen käyttäen Pilvilammen omaa Sopivin 365 sovellusalustaa. Järjestelmän tarkoitus oli työnkulun helpottaminen sekä asiakkaan tietojen vaivaton ylläpito SharePoint-ympäristössä. Järjestelmä tekee asiakkaan työnteosta merkittävästi helpompaa ja järjestelmällistä.

Lopputuloksena syntyi helppokäyttöinen ja tyylikäs sivusto, joka täytti kaikki asiakkaan tämänhetkiset vaatimukset. Sivustosta tuli loppujen lopuksi melko iso mutta sen käytettävyys pysyi silti erinomaisena. Etenkin dokumenttien generoinnit saatiin nopeutettua ja optimoitua loistavalle tasolle.

Sivusto on luovutettu asiakkaan käyttöön ja siitä on saatu palautetta. Sivustosta voi löytyä bugeja ja ongelmia asiakkaan testauksessa, joita korjataan pois sitä mukaan, kun niitä tulee vastaan. Asiakkaan on helppo olla yhteydessä sivustolla sijaitsevan helpdesk-sovelluksen kautta. Tällä tavoin kaikki ongelmat saadaan korjattua pois mahdollisimman nopeasti ja mahdollista jatkokehitystä voidaan alkaa toteuttamaan.

Suurimmat ongelmat projektissa tulivat vastaan Power Automate-osuudessa, jolloin jouduttiin luomaan paljon monimutkaisia ehtoja vaativia dokumentteja. SharePoint- ja AngularJS-osuudet eivät tuottaneet ongelmia vaan olivat melko yksinkertaisia. Projekti pysyi hyvin aikataulussa mutta resurssienhallinnassa olisi ollut hieman parannettavaa.

Projekti vahvensi osaamistani SharePoint- ja AngularJS-tekniikoissa, joita olin jo ehtinyt opettelemaan hyvän aikaa ennen projektin aloittamista. Tämä projekti oli kuitenkin ensimmäisiä kertoja, kun pääsin kunnolla kokeilemaan Power Automate-sovelluksen toimintaa. Projekti opetti paljon Power Automaten toiminnasta ja etenkin sen sisäisten ehtojen ja muuttujien käytöstä.

LÄHTEET

Alexander, M. 2016. Excel Power Pivot & Power Query For Dummies. For Dummies; 1st edition. Viitattu 29.3.2021.

AngularJS. 2021a. API: ngRoute. Viitattu 9.10.2021. https://docs.angularjs.org/api/ngRoute

AngularJS. 2021b. Developer Guide: Data Binding. Viitattu 19.9.2021. https://docs.angularjs.org/guide/databinding

AngularJS. 2021c. Developer Guide: Controllers. Viitattu 26.9.2021. https://docs.angularjs.org/guide/controller

AngularJS. 2021d. Developer Guide: Dependency Injection. Viitattu 19.9.2021. https://docs.angularjs.org/guide/di

AngularJS. 2021e. Developer Guide: Directives. Viitattu 18.9.2021. https://docs.angularjs.org/guide/directive

AngularJS. 2021f. Developer Guide: E2E Testing. Viitattu 9.10.2021. https://docs.angularjs.org/guide/e2e-testing

AngularJS. 2021g. Developer Guide: Expressions. Viitattu 25.9.2021. https://docs.angularjs.org/guide/expression

AngularJS. 2021h. Developer Guide: Filters. Viitattu 6.10.2021. https://docs.angularjs.org/guide/filter

AngularJS. 2021i. Developer Guide: Forms. Viitattu 3.10.2021. https://docs.angularjs.org/guide/forms

AngularJS. 2021j. Developer Guide Introduction: What is AngularJS? Viitattu 2.8.2021. https://docs.angularjs.org/guide/introduction

AngularJS. 2021k. Developer Guide: Providers. Viitattu 14.10.2021. https://docs.angularjs.org/guide/providers

AngularJS. 2021l. Developer Guide: Services. Viitattu 6.10.2021. https://docs.angularjs.org/guide/services

AngularJS. 2021m. Developer Guide: Unit Testing. Viitattu 9.10.2021. https://docs.angularjs.org/guide/unit-testing

AngularJS. 2021n. Developer Guide Scopes: What are Scopes? Viitattu 26.9.2021. https://docs.angularjs.org/guide/scope

AngularJS. 2021o. Tutorial: Routing & Multiple Views. Viitattu 9.10.2021. https://docs.angularjs.org/tutorial/step_09

Formly. 2022. Viitattu 30.1.2022. https://formly.dev/

Green, B. & Seshadri, S. 2014. AngularJS: Up and Running. O'Reilly Media; 1st edition (October 7, 2014).

Guilmette, A. 2020. Workflow Automation with Microsoft Power Automate: Achieve digital transformation through business automation with minimal coding. E-kirja. Packt Publishing.

Joseph, A. 2021. How to Create A Document Library in SharePoint. Viitattu 12.7.2021. [blogikirjoitus]. https://blog.mydock365.com/how-to-create-a-document-library-in-sharepoint

Lerner, A. 2013. ng-book – The Complete Book on AngularJS. Fullstack io; 1st edition.

Mc Dermid, C. 2018. SharePoint Metadata – What Is It and Why Is It Important? Viitattu 11.9.2021. [blogikirjoitus]. https://miktysh.com.au/sharepoint-metadata-what-is-it-and-why-is-it-important/

Microsoft. 2021a. Create a cloud flow from a template in Power Automate. Viitattu 7.11.2021. https://docs.microsoft.com/en-us/power-automate/get-started-logic-template

Microsoft. 2021b. Create, change, or delete a view of a list or library. Viitattu 1.8.2021. https://support.microsoft.com/en-us/office/create-change-or-delete-a-view-of-a-list-or-library-27ae65b8-bc5b-4949-b29b-4ee87144a9c9

Microsoft. 2021c. Create list relationships by using unique and lookup columns. Viitattu 11.12.2021. https://support.microsoft.com/en-us/office/create-list-relationships-by-using-unique-and-lookup-columns-80a3e0a6-8016-41fb-ad09-8bf16d490632

Microsoft. 2021d. Data protection in connectors. Viitattu 7.11.2021. https://docs.microsoft.com/en-us/connectors/protection

Microsoft. 2021e. List and library column types and options. Viitattu 11.6.2021. https://support.microsoft.com/en-us/office/list-and-library-column-types-and-options-0d8ddb7b-7dc7-414d-a283-ee9dca891df7#ID0EAABAAA=Modern

Microsoft. 2021f. RESTful web API design. Viitattu 11.12.2021. https://docs.microsoft.com/fi-fi/azure/architecture/best-practices/api-design

Microsoft. 2021g. Use expressions in condition to check multiple values. Viitattu 7.11.2021. https://docs.microsoft.com/en-us/power-automate/use-expressions-in-conditions

Microsoft. 2021h. What is an on-premises data gateway? Viitattu 7.11.2021. https://docs.microsoft.com/en-us/power-automate/gateway-reference

Niaulin, B. 2014. What are SharePoint Content Types? Learn How to Create and Use Them. Viitattu 1.8.2021. [blogikirjoitus]. https://sharegate.com/blog/sharepoint-content-types-understand-use-create

Plumsail. 2022a. Create documents from template. Viitattu 30.1.2022. https://plumsail.com/documents/create-documents-from-templates

Plumsail. 2022b. Introduction to building online forms with Plumsail Forms. Viitattu 30.1.2022. https://plumsail.com/docs/forms-web/introduction.html

Reinders, M. 2021. Microsoft Lists – An Evolution of SharePoint Lists. Viitattu 1.8.2021. https://petri.com/microsoft-lists-an-evolution-of-sharepoint-lists

tutorialspoint. 2021a. MVC Framework – Introduction. Viitattu 8.9.2021. https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm

tutorialspoint. 2021b. SharePoint – Libraries. Viitattu 12.9.2021. https://www.tutorialspoint.com/sharepoint/sharepoint_libraries.htm

W3 Schools. 2021a. AngularJS Expressions. Viitattu 25.9.2021. https://www.w3schools.com/angular/angular_expressions.asp

W3 Schools. 2021b. AngularJS Filters. Viitattu 20.10.2021. https://www.w3schools.com/angular/angular_filters.asp

Zelfond, G. 2020. How to create a Lookup column in SharePoint. Viitattu 11.6.2021. https://sharepointmaven.com/how-to-create-a-lookup-column-in-sharepoint/