HAMK
Hämeen ammatti-
korkeakoulu

SOVELLUSKEHITYS SAP BUILD CODELLA JA JOULELLA

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus

Kevät 2024

Minttu Laesmaa

HAMK
Hämeen ammatti-
korkeakoulu

Tietojenkäsittelyn koulutus Tiivistelmä
Tekijä Minttu Laesmaa Vuosi 2024
Työn nimi Sovelluskehitys SAP Build Codella ja Joulella
Ohjaaja Lasse Seppänen

Opinnäytetyön tavoitteena oli tutkia, miten SAP Build Code ja Joule voivat vaikuttaa ohjelmistokehityksen tehokkuuteen ja uusien kehittäjien perehdyttämiseen SAP-ympäristössä. Tutkimus keskittyi arvioimaan työkalujen vahvuuksia ja heikkouksia erityisesti generatiivisen tekoälyn osalta ja se suoritettiin toimeksiantajayrityksen sovellukseen pohjautuvan kehitysprojektin kautta. Opinnäytetyön toimeksiantajana on suomalainen SAP-teknologioihin erikoistunut konsultointiyritys.

Opinnäytetyön teoreettinen viitekehys käsittelee SAP:ia, SAP Build -alustaa sekä generatiivisen tekoälyn roolia yritysmaailmassa. Opinnäytetyö on toiminnallinen, ja työkalujen arviointi tehtiin käytännön sovelluskehitysprojektin yhteydessä.

Tutkimuksessa analysoitiin SAP Build Coden ja Joulen suorituskykyä SAPUI5-kehityksessä usein toistuvien sovelluskehitystehtävien kautta. Johtopäätöksissä todettiin, että työkalut voivat olla merkittävä apu sovelluskehityksen parissa, vaikka niiden käyttö vaatiikin vielä tarkkuutta ja teknistä osaamista. Tulokset osoittavat, että SAP Build Code ja Joule voivat rajallisesti toimia hyödyllisinä apuvälineinä uusille kehittäjille, joilla on kuitenkin jo tietämystä SAPUI5-sovelluskehityksestä.

Avainsanat SAP, SAP Build Code, Joule, sovelluskehitys, generatiivinen tekoäly
Sivut 57 sivua ja liitteitä 1 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Degree Programme in Business Information Technology — Abstract
Author Minttu Laesmaa — Year 2024
Subject Software development with SAP Build Code and Joule
Supervisors Lasse Seppänen

The purpose of the thesis was to investigate how SAP Build Code and Joule can affect the efficiency of software development and the introduction of new developers in an SAP environment. The research focused on assessing the strengths and weaknesses of the tools, especially regarding generative artificial intelligence, and was conducted through a development project based on the commissioner's application. The thesis was commissioned by a Finnish consulting company specialising in SAP technologies.

The theoretical framework of the thesis deals with SAP, SAP Build platform and the role of generative artificial intelligence in the business world. The thesis is practical, and the tools were evaluated in the context of a concrete application development project.

The study analyzed the performance of SAP Build Code and Joule in SAPUI5 development through frequently occurring application development tasks. It was concluded that the tools can be of significant help in application development, although their use still requires precision and technical knowledge. The results show that SAP Build Code and Joule have limited potential as useful tools for new developers who already have some knowledge of SAPUI5 application development.

Keywords SAP, SAP Build Code, Joule, software development, generative artificial intelligence
Pages 57 pages and appendices 1 pages

Sanasto

SAP	Saksalainen ohjelmistoalan yritys
BTP	Business Technology Platform
BAS	Business Application Studio
AI	Tekoäly (Artificial Intelligence)
GAI	Generatiivinen tekoäly (Generative Artificial Intelligence)
MVC	Ohjelmistoarkkitehtuurin suunnittelumalli (Model-View-Controller)
UI	Käyttöliittymä (User Interface)
UX	Käyttökokemus (User Experience)
XML	Tiedostomuoto tietojen rakenteiden esittämiseen (Extensible Markup Language)
JSON	Tiedot avain-arvo-pareina säilyttävä tiedostomuoto (JavaScript Object Notation)
Unkarilainen notaatio	Nimeämiskäytäntö, jossa muuttujan nimi alkaa sen tyyppiä tai käyttötapaa ilmaisevalla pienellä kirjaimella

Sisällys

Kuvat, komennot, ohjelmakoodit, taulukot ja kaavat

Liitteet

1 Johdanto

Sovelluskehitys on jatkuvasti kehittyvä ala, jossa uudet teknologiat tarjoavat sekä mahdollisuuksia että haasteita ohjelmistokehittäjille. SAP Build Code ja Joule edustavat näistä teknologioista tuoreinta suuntausta, tarjoten työkaluja tehokkaiden ja käyttäjäystävällisten sovellusten rakentamiseen. Tämä opinnäytetyö pyrkii selvittämään, minkälaista tukea sovelluskehittäjän on mahdollista saada SAP:n omasta generatiivisesta tekoälystä, voisiko työkaluista olla avuksi uusien SAP-kehittäjien perehdyttämisessä ja kuinka työkalut vaikuttavat yleisesti sovelluskehitysprojektin tehokkuuteen ja sujuvuuteen.

SAP Build Codea ja Joulea tullaan kokeilemaan kehittämällä yksinkertainen, toimeksiantajayrityksen olemassa olevaan sovellukseen perustuva UI-käyttöliittymä, jossa käyttäjä pystyy hallinnoimaan kahdessa eri taulussa olevia tietoja. Opinnäytetyössä keskitytään tarkastelemaan, kuinka hyvin valitut työkalut kokonaisuudessaan selviytyvät kehitysprosessista.

Valmiin opinnäytetyön tuloksilla tulee olemaan keskeinen rooli ohjelmistokehitysyritysten päätöksenteossa SAP-ympäristössä. Tuloksia voidaan käyttää apuna päätöksenteossa siinä, voidaanko yrityksen sisällä hyötyä SAP Build Coden ja Joulen käyttöönotosta.

Opinnäytetyössä tullaan vastaamaan seuraaviin tutkimuskysymyksiin:

1. Miten SAP Build Code ja Joule vaikuttavat kehitysprosessin tehokkuuteen ja sujuvuuteen?

2. Mitkä ovat Joulen heikkoudet ja vahvuudet kehitystyön apulaisena?

3. Voidaanko Joulea käyttää uusien SAP-kehittäjien perehdyttämisessä?

2 SAP

Saksassa vuonna 1972 perustettu SAP on yksi maailman johtavista liiketoimintaprosessien hallintaan tarkoitettujen ohjelmistojen tuottajista. SAP on lyhenne sanoista Systemanalyse programmentwicklung (System Analysis Program Development), joka oli yrityksen alkuperäinen nimi. (SAP, ei pvm.-k)

Alun perin viiden entisen IBM:n työntekijän perustama pienyritys on kasvanut monikansalliseksi yritykseksi, joka työllistää tänä päivänä yli 105 000 työntekijää. Sillä on yli 230 miljoonaa pilviasiakasta, yli 100 eri liikentoimintaprosesseja kattavaa ratkaisua ja kaikista palveluntarjoajista suurin pilvipalveluvalikoima. (SAP, ei pvm.-i)

2.1 SAP Build

Vuonna 2021 kansainvälinen tieto- ja viestintäteknologian tutkimus- ja konsultointiyritys Gartner ennusti organisaatioiden tulevan kehittämään 70 % uusista sovelluksistaan hyödyntäen vähäkoodisia sovelluskehitysteknologioita (low-code/no-code), kun vuonna 2020 niitä käytti vain alle 25 %. Vähäkoodisten teknologioiden suosion nousua on ajanut monista eri tekijöistä muun muassa kasvava tarve nopeaan ja kustannustehokkaaseen sovelluskehitykseen, pilvipohjaisten palveluiden suosion kasvu sekä niiden mukanaan tuomat uudet työnkuvat sekä vastuualueet. (Gartner, ei pvm.) Vähäkoodiset teknologiat tarjoavat uuden tavan suunnitella ja kehittää ohjelmistoja hyödyntäen graafisia työkaluja ja sulautettuja toimintoja, jotka vähentävät tai poistavat kokonaan tarpeen kirjoittaa perinteistä koodia (SAP, ei pvm.-b).

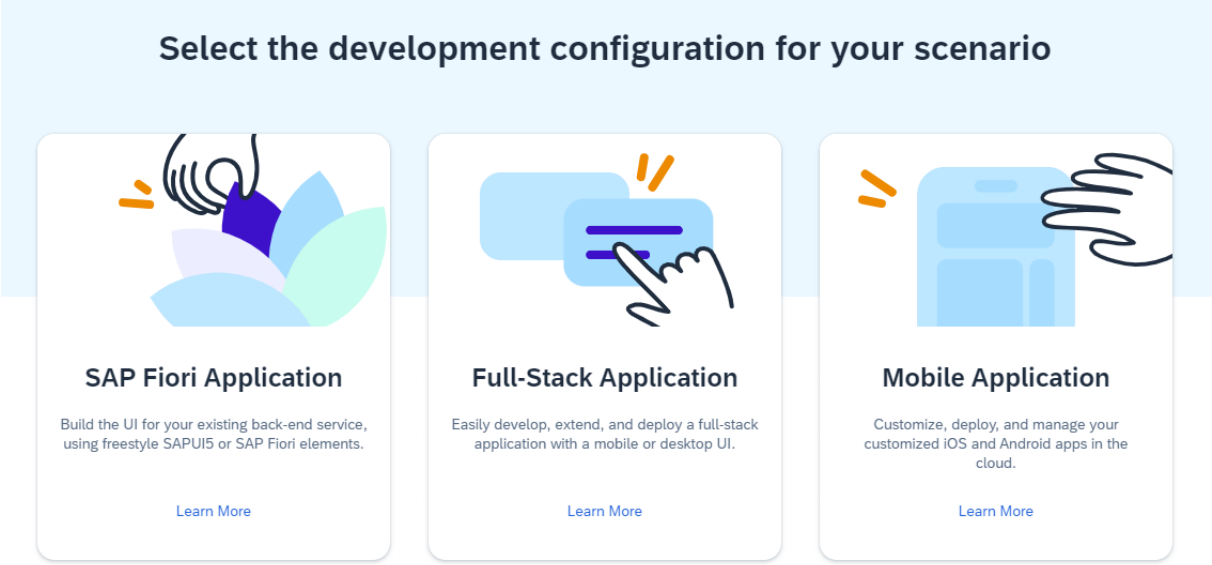
SAP Build on vuonna 2023 julkaistu työkaluportfolio, joka on suunniteltu yksinkertaistamaan ja nopeuttamaan sovellusten ja automaatioiden kehittämistä. Julkaisuhetkenään se sitoi kolme aiemmin itsenäistä sovellusta yhtenäiseksi kehitysympäristöksi. Vähäkoodiseen kehitykseen erikoistuneet työkalut SAP Build Apps, SAP Build Process Automation ja SAP Build Work Zone ovat suunniteltu mahdollistamaan yrityssivustojen suunnittelun ja prosessien automatisoinnin ilman suurta kehitystiimiä tai teknistä asiantuntemusta. (Gupta, 2023; PRESS, ei pvm.)

2.2 SAP Build Code ja Joule

SAP Buildin uudeksi työkaluksi maaliskuussa 2024 julkaistu SAP Build Code on ns. ”pro-code”-työkalu, joka tarjoaa valmiin sovelluskehitysympäristön hyödyntämällä generatiivista Joule-tekoälyä. Joulen avulla voidaan luoda koodia ja sovelluslogiikkaa SAP:n ohjelmointimallien mukaisesti luonnollisilla kielillä tehdyistä kehotteista ja tuottaa nopeasti yksikkötestejä laadun ja tarkkuuden lisäämiseksi. SAP-kehitykseen räätälöitynä Build Coden luvataan myös tekevän esimerkiksi valmiiden integraatioiden ja asiakasrajapintojen hyödyntämisen helpoksi, sekä mahdollistavan SAP BTP:n ominaisuuksia todennuksessa, auktorisoinnissa ja SAP-tietojen suojaamisessa. (SAP, ei pvm.-h)

SAP Build Code tarjoaa ennalta konfiguroidun kehitysympäristön SAP Fiori -sovelluksille, fullstack-sovelluksille ja mobiilisovelluksille (Kuva 1). SAP Build Coden kehitysympäristössä käyttäjä voi valita työskentelytapansa: kehitystyötä voi tehdä perinteistä koodia kirjoittamalla tai vaihtoehtoisesti erilaisten SAP Build Coden tarjoamien visuaalisten editorien ja työkalujen avulla. (SAP Community, 2024)

Kuva 1 SAP Build Coden tarjoamat kehitysympäristöt



SAP Business Application Studio on Microsoft Visual Studio Codeen perustuva sovelluskehitystyökalu (IDE), jossa on hyvin samanlainen visuaalinen käyttöliittymä kuin esikuvassaan. Visual Studio Coden lailla siihen on myös mahdollista kehittää omia lisäosiaan ja laajennuksiaan, joita voidaan hyödyntää kehitysympäristöissä. (SAP Community, 2024)

2.2.1 Julkaisuhetki

SAP Build Coden julkaisupäivänä SAP järjesti videopalvelu YouTubessa SAP Build Code – Ask Me Anything -livelähetyksen, jossa demottiin SAP Build Coden sisältämän generatiivisen tekoälyn, Joulen, kykyä muuttaa luonnollisen kielen kehote koodiksi ja annettiin osallistujille mahdollisuus kysyä uusiin työkaluihin liittyviä kysymyksiä. (SAP Community, 2024)

Lähetyksessä esitettiin kysymys Joulen kyvystä muokata tai parannella olemassa olevaa koodia. Kysymykseen vastanneen Yuval A.:n (Lead Product Owner of SAP Business Application Studio and SAP Build Code developer) mukaan Joulea pystyi toistaiseksi hyödyntämään vain uuden koodin tuottamiseen. Mahdollisuuden muokata olemassa olevaa koodia luvattiin kuitenkin olevan kehityksessä korkealla prioriteetilla. (SAP Community, 2024)

Lähetyksessä kysyttiin myös, onko Joule tietoinen kaikista projektin tiedostoista vai ottaako se vastauksia ja koodia generoidessaan huomioon ainoastaan sen, mitä käyttäjä on komentokehotteeseensa kirjoittanut. Tim B:n mukaan kysymykseen ei voinut vastata ysinkertaisesti joko-tai, sillä vaikka Joule yleisesti ottaen oli tietoinen kaikista projektin tiedostoista, oli vielä joitakin osia, joita se ei vielä huomioinut lainkaan. Vastauksessaan Tim B. myös korosti, että Joulen oli oltava tietoinen projektin tiedostoista, jotta se osaisi generoida esimerkiksi tietokannan entiteettien mukaisia rakenteita. (SAP Community, 2024)

Tilaisuudessa kerrottiin kehittäjiltä kysyttävän usein, että mikä tekee Joulesta yleisesti ottaen muuta generatiivista tekoälyä paremman. Tähän vastattiin, että Joule on tietoinen koko projektista, teknologiasta sekä kontekstista, jonka vuoksi se osaa tuottaa käyttäjälleen SAP:n standardien mukaisia ratkaisuja esitettyihin ongelmiin. Vaikka Joulen vastausgenerointi edellyttää tietoisuutta projektin tiedostoista, ei käyttäjien tietoja luovuteta Joulelle, eikä tekoälymallia kouluteta käyttäjien tiedoilla. (SAP Community, 2024)

2.2.2 Joulen tulevaisuus

Joulen tulevaisuuden suunnitelmia avattiin kesäkuussa 2024 pidetyssä SAP Sapphire -tapahtumassa, jossa ilmoitettiin Joulen integroimisesta Microsoftin rinnalle Microsoft 365 -ympäristöön. 2024 loppuvuoden aikana alkavan integraation luvattiin yhdistävän SAP-järjestelmissä olevat yritystiedot esimerkiksi Microsoft Teamsin ja Microsoft Outlookin tietoihin, mikä tulee mahdollistamaan entistä paremmat tiedot päätöksenteon parantamiseksi. (Herzig, 2024)

Samassa tilaisuudessa julkistettiin myös Joulelle kehityksessä olevat konsultointia ja ABAP-kehitystä tukevat ominaisuudet, joiden mainostettiin nopeuttavan projektien ja SAP-ratkaisujen toteuttamista 30 %. Tulevaisuudessa Joulen myös suunniteltiin kuuluvan kaikkiin pilviasiakkaille tarjottaviin ratkaisuihin. Uskottiin, että vuoden 2024 loppuun mennessä yleisimpien tehtävien hallinnasta jopa 80 % suoritettaisiin Joulen avulla. (Taylor, 2024)

3 SAP Fiori & SAPUI5-sovelluskehitys

SAPUI5:n kehitystyö alkoi vuonna 2008, jolloin se tunnettiin vielä nimellä Project Phoenix, mutta todellinen suosion kasvu alkoi toukokuussa 2013, kun SAP Fiori julkaistiin (SAP, 2019). SAPUI5:n kehityksessä on keskitytty neljään avainalueeseen, jotka ovat modulaarinen ydin, renderöinti ja ohjaimet, ohjelmointimallit ja työkalujen rakentaminen. Modulaarisella ytimellä SAPUI5 parantaa koodin organisointia ja riippuvuuksien hallintaa, mikä minimoi sovellusten käyttämät resurssit. Käyttökokemus muuttuu kevyemmäksi ja nopeammaksi kompaktin koodikokonaisuuden ansiosta. Renderöinnin ja ohjaimien osalta pyritään yksinkertaistamaan renderöintiprosessia sekä luomaan pieniä ja suljettuja ohjaimia. SAPUI5:n ohjelmointimallia jatkokehitetään edelleen parantamaan yhteensopivuutta muiden suosittujen kehysratkaisujen, kuten Angularin ja Reactin, välillä. (Goebels ym., 2020, s. 27)

SAP Fiori on SAPUI5-teknologiaan perustuva sovelluskehys, joka esiteltiin ensimmäisen kerran vuonna 2013. SAP Fiori tarjoaa valikoiman valmiiksi konfiguroituja sovelluksia erilaisille liiketoiminta-alueille, kuten henkilöstöhallinnolle, ostopalveluihin ja myyntiin. Se varmistaa, että eri laitteilla sovellukset ovat yhdenmukaisia ja tarjoaa SAP-sovelluksille modernin, käyttäjäystävällisen ja intuitiivisen käyttöliittymän. (Glavanovits ym., 2023, ss. 58–59)

SAP Fiori ja SAPUI5 liittyvät monin tavoin toisiinsa, minkä vuoksi näitä termejä käytetään usein ristiin. Tästä huolimatta ne eivät ole millään tavoin toisistaan teknisesti riippuvaisia, vaan sovelluksia voidaan kehittää myös ilman toista työkalua. (Glavanovits ym., 2023, s. 38; Goebels ym., 2020, s. 53)

3.1 MVC-suunnittelumalli

Sovelluksen esitys- ja logiikkakerrosten eriyttämiseen suunniteltu Model-View-Controller (MVC) on perinteinen ohjelmistokehityksen suunnittelumalli. Sitä hyödyntämällä ohjelmiston rakenne jaetaan kolmeen, erilliseen osa-alueeseen, jotka ovat kuitenkin vuorovaikutuksessa keskenään ja usein myös toisistaan riippuvaisia. MVC-rakenne koostuu seuraavista osista:

1. Malli (model) säilyttää ja hallinnoi sovelluksessa käytettäviä tietorakenteita
2. Näkymä (view) vastaa käyttäjälle esitettävää, visuaalista käyttöliittymää
3. Ohjain (controller) vastaanottaa tapahtumia ja komentoja näkymän ja mallin välillä. (hurjatron, 2020)

SAPUI5 sisältää kolme käyttövalmista, joita ovat oData-malli, JSON-malli, XML-malli ja resurssimalli. Opinnäytetyössä kehitettävässä sovelluksessa tullaan käyttämään JSON-mallia, joka on suunniteltu pienien tietojoukkokokonaisuuksien hallinnoimiseen. Malli on ns. asiakaspuolen malli, jossa sovelluksen tiedot ladataan kokonaisuudessaan asiakkaan käytettäviksi. Myös erilaiset tietojen hallintaoperaatiot, kuten suodatus ja uudelleenjärjestys, suoritetaan ilman ylimääräisiä palvelinpyyntöjä. (SAP, ei pvm.-g)

Lisäksi sovelluksessa tarvittavien tekstien, kuten syöteotsikoiden ja sarakkeiden nimien, hallinnoimiseen käytetään resurssimallia. Resurssimalli mahdollistaa esimerkiksi sovelluksen kielen lokalisoinnin käyttäjän valitseman kielen mukaisesti. (SAP, ei pvm.-g)

Näkymät toteutetaan SAPUI5:n suositusten mukaisesti XML-näkyminä, jotka edistävät koodin luettavuutta ja ylläpidettävyyttä. XML-näkymät voidaan koostaa eri SAPUI5-kirjastojen ohjainelementeistä, kuten taulukoista, painikkeista ja syötteistä. (SAP, ei pvm.-m)

3.2 Ohjainelementit

SAPUI5-sovelluskehityksessä käyttöliittymien ulkoasu ja funktionaalisuudet rakennetaan ohjainelementeillä (control), joita ovat esimerkiksi painike (sap.m.Button) tai responsiivinen työkalurivi (sap.m.OverflowToolbar) (SAP, ei pvm.-l, ei pvm.-c, ei pvm.-j).

Kunkin ohjainelementin metatiedoissa määritetään sen ominaisuudet, tapahtumat, koosteet sekä yhteydet. Metatieto toimii julkisena rajapintana, jota ohjainelementtiä käyttävät sovellukset voivat hyödyntää. Ominaisuudet, kuten ohjainelementin teksti tai leveys, vaikuttavat esimerkiksi sen ulkoasuun tai sen esittämiin tietoihin käyttöliittymässä. Ohjainelementin koosteet (aggregation) toimivat säiliöinä, joiden avulla ohjainelementin sisälle voidaan määrittää esimerkiksi joukko pääohjainelementistä riippuvaisia lapsiohjainelementtejä. (SAP, ei pvm.-l)

XML-näkymään luotavan ohjainelementin lapsiohjainelementeille ei erikseen tarvitse määrittää mihin koosteeseen ne kuuluvat, mikäli ne on tarkoitus sijoittaa pääohjainelementin vakiokoosteen sisälle. Mikäli ohjainelementit halutaan sijoittaa johonkin muuhun koosteeseen tai pääohjainelementille ei ole asetettu vakiokoostetta, tulee kooste merkitä XML-näkymään. (SAP, ei pvm.-a)

Ohjelmakoodi 1 sisältää yksinkertaisen esimerkin ohjainelementtien koosteiden käytöstä XML-näkymässä. List-ohjainelementin items-parametriin on määritetty polku, jossa sijaitsee items-koosteeseen näkyviin tulevat tietueet. Jotta tiedot näkyvät käyttöliittymässä halutussa muodossa, tulee pääohjainelementin sisälle määrittää myös lapsiohjainelementti. Esimerkissä käytettävällä sap.m.List-ohjainelementillä voidaan esittää erimuotoista listattua tietoa, johon SAPUI5 tarjoaa valmiita List Item-ohjainelementtejä. Näitä ovat esimerkiksi syötemuotoiselle tiedolle tarkoitettu Feed List Item ja yksinkertaisemmalle tiedolle tarkoitettu Standard List Item. (SAP, ei pvm.-f)

Ohjelmakoodi 1 Esimerkki pää- ja lapsiohjainelementeistä

```
<List
      id=”companyList”
      items="{
           path: '/companies',
           templateShareable:false
           }"
>
           <items>
                      <StandardListItem
                                 title="{name}"
                                 description="{city}"
                      />
           </items>
</List>
```

3.3 Tietojen sitominen

Tietojen sitomisen tavalla sovelluksessa hallinnoidaan mallien ja näkymien keskinäistä kommunikointia. Sovelluksissa käytettävissä olevat tiedonsidontatavat riippuvat käytettävästä mallista. (SAP, ei pvm.-d)

Opinnäytetyön projektissa tietojoukkojen hallinnoimiseen käytettävä JSON-malli tukee kaikkia kolmea tietojen sidonnan mallia, joita ovat kertasidonta, yksisuuntainen sidonta ja kaksisuuntainen sidonta. Se käyttää oletuksena kaksisuuntaista sidontaa, joka mahdollistaa näkymässä muutettujen tietojen tallentamisen malliin ja mallissa muutettujen tietojen päivittämisen näkymään. Tekstitiedot säilyttävä resurssimalli tukee kertasidonnan lisäksi vain yksisuuntaista sidontaa, jota se myös käyttää oletusarvoisesti. Koska tekstitietoja ei ole tarkoitus voida muuttaa esimerkiksi käyttöliittymänäkymän kautta, tarvitaan tietojen näyttämiseen vain yksisuuntainen sidonta mallista näkymään. (SAP, ei pvm.-d)

Malliin ladatut tiedot voidaan sitoa näkymän ohjainelementteihin kolmella eri käyttötarkoituksesta riippuvaisella tavalla:

1. Ominaisuuksien sidonnalla ohjainelementin ominaisuudet päivittyvät automaattisesti malliin tallennettujen tietojen mukaisesti. Kontekstisidonnasta poiketen ominaisuussidonnassa viitataan objektin pääavaimen sijasta suoraan sidottavan ominaisuuden avaimeen
2. Kontekstisidonnalla näkymän ohjainelementti voidaan sitoa mallin objektiin, jonka jälkeen objektin sisällä olevia elementtejä voidaan hyödyntää ilman pääavainta
3. Listasidonnan avulla näkymään voidaan luoda ohjainelementtien lapsielementtejä (esimerkiksi taulukon sarakkeet ja rivit) automaattisesti malliin tallennettujen tietojen pohjalta. Lapsielementtien luomiseen voidaan käyttää esimerkiksi SAPUI5:n tehdasfunktiota. (SAP, ei pvm.-b)

4 Generatiivinen tekoäly yritysmaailmassa

Generatiivisella tekoälyllä viitataan tekoälymalleihin, joiden tarkoitus on luoda esimerkiksi kirjallisen tekstin, äänen, kuvien tai videon muodossa olevaa uutta sisältöä. Se voidaan asettaa esimerkiksi imitoimaan valitun kirjailijan tai säveltäjän tyyliä, herättämään tekstimuotoinen kuvaus eloon videon muodossa tai luoda potretti henkilöstä, jota ei ole olemassa. (SAP, ei pvm.-e)

4.1 Yritysmaailman suhtautuminen tekoälyyn

Deloitten vuonna 2022 teettämässä tutkimuksessa 2620 haastatellusta yritysjohtajasta 94 % prosenttia vastaajista uskoi tekoälyn hyödyntämisen olevan tärkeä osa yritysten kasvua. Toisaalta vaikka 82 % uskoi tekoälyn hyödyntämisen parantavan alaistensa tehokkuutta sekä työviihtyvyyttä, olivat tekoälyn käytön edistämistä koskevien kysymysten prosenttiluvut päinvastaiset. Vastaajista 75 % ei tarjonnut työntekijöilleen käyttäjäystävällisiä tekoälyjärjestelmiä, 70 % ei ottanut työntekijöitä mukaan tekoälyn suunnitteluun ja jopa 79 % prosenttia ei kouluttanut työntekijöitään tekoälyn tehokkaaseen käyttöön. (Deloitte, 2022, ss. 5, 15, 17)

Yritykset tuntuivat yleisesti ottaen suosivan valmiita ratkaisuja: tekoälyyn erikoistuvia työntekijöitä mieluummin palkattiin, kuin koulutettiin nykyistä työvoimaa ja oman tuotteen kehittämisen sijaan tekoälyratkaisut ostettiin mieluiten valmiina palveluna (Deloitte, 2022, s. 25).

Suomessa vastaavanlaista tietoa tekoälyä käyttävistä yrityksistä on kerätty Solitan ja IRO Researchin vuonna 2023 teettämässä kyselyssä, johon kerättiin vastauksia Suomen 500 suurimman yrityksen johtotehtävissä työskenteleviltä henkilöiltä. Tekoälyn käyttöönotto Suomessa on ollut kansainvälisiin markkinoihin nähden verrattain hidasta. Jopa 70 % prosenttia vastaajista ei käyttänyt generatiivista tekoälyä työnsä tukena edes ajoittain, mikä kuvastaa varovaista suhtautumista muutokseen. Kansainvälisten trendien kanssa ristiriidassa oli myös vastaajien näkemys siitä, kuinka paljon generatiivinen tekoäly tulee vaikuttamaan organisaatioiden toimintaan: vastaajista 54 % uskoi vaikutuksia olevan vain vähän tai ei lainkaan. Kuitenkin yrityksissä, joissa generatiivinen tekoäly oli jo otettu käyttöön, suhtautuminen oli huomattavasti positiivisempaa ja jopa 75 % piti generatiivisen tekoälyn vaikutuksia merkittävinä. (Solita & IRO Research, 2023, ss. 4–6)

4.2 Riskit generatiivisen tekoälyn käyttöönotossa

Deloitten tuottaman kyselyn mukaan tekoälyn käyttöönoton haasteista nousi esiin esimerkiksi seuraavia seikkoja: tekoälyn riskitekijöiden hallinta, yritysjohdon sitoutumisen puute ja ylläpidon tai teknisen tuen puutteet. Vastaukset korostivat selkeän johtajuuden ja investointien merkitystä tekoälyyn liittyvissä muutoksissa, ja ovat toistuneet Deloitten tutkimuksien aiemmissakin versioissa. (Deloitte, 2022, ss. 5, 8) Lisäksi vastaajista vain 33 % prosenttia oli sovittanut tekoälyyn liittyvien riskien hallinnan osaksi organisaationsa laajempia riskienhallintatoimia, vaikka 50 % oli nostanut ne yhdeksi merkittävimmistä esteistä tekoälyprojektien skaalaamiselle. Luku selittyy kuitenkin osittain ulkopuolisilla palveluntarjoajilla teetetyillä tietoturva-auditoinneilla. (Deloitte, 2022, ss. 15, 17, 22)

Turvallisuus- ja yksityisyysriskit, taitojen puute, väärinkäyttö ja saatavilla olevan tiedon laatu olivat myös suomalaisten yritysjohtajien nimeämiä seikkoja generatiivisen tekoälyn suurimmiksi riskeiksi. Vaikka osa vastaajista näkikin generatiivisen tekoälyn mahdollisuutena vahvistaa yrityksensä tietoturvaa, jopa 74 % vastaajista koki generatiivisen tekoälyn käyttöönoton aiheuttavan turvallisuus- ja yksityisyysriskejä liiketoiminnalleen. (Solita & IRO Research, 2023, s. 15)

5 Sovelluksen kehitys- ja työkalujen testausprosessi

Tässä luvussa esitellään opinnäytetyössä käytetyt projektinhallintamenetelmät, miten käytännön sovellusprojekti on suunniteltu ja toteutettu, sekä mitkä ovat olleet opinnäytetyön lopputavoitteet.

5.1 Projektimenetelmät

Opinnäytetyön projektinhallintatyökaluna käytetään Microsoft Planner -työkalua ja Kanban-metodia. Plannerin työtilaan luodaan omat tehtäväsarakkeensa opinnäytetyössä käsiteltäville teemoille ja luvuille. Toteutettavat työvaiheet pilkotaan sarakkeiden sisälle omiksi tehtäväkorteikseen, jotka edistävät opinnäytetyön rakenteen suunnittelua ja ylläpitoa.

Työssä tarkastellaan ja pohditaan Joulen heikkouksia ja vahvuuksia konkreettisen sovelluskehitysprojektin kautta. Kehitystyön yhteydessä tehdyt havainnot kirjataan ylös kuvankaappauksina sekä tekstimuotoisina muistioina, jotka kirjoitetaan puhtaaksi ja avataan opinnäytetyön käytäntöä koskeviin lukuihin.

Kehitettävä sovellus pohjautuu toimeksiantajayrityksen olemassa olevalle projektille ja siinä käytetään valmista projektipohjaa. Näiden vuoksi opinnäytetyössä ei keskitytä itse sovelluksen suunnitteluprosessiin, vaan päästään keskittymään työkalujen suoriutumiseen ja tutkimaan Joulen kykyä toimia sovelluskehityksen apurina.

5.2 Suunnittelu ja toteutus

Opinnäytetyössä toteutettavan käytännön sovelluskehitysprojektin suunnittelu aloitettiin jo ennen työkalujen varsinaista julkaisua seuraamalla aktiivisesti SAP:n julkaisemia uutisia ja tiedotteita SAP Build Codea ja Joulea koskien. Työkaluista annettuihin lupauksiin tutustuminen auttoi saamaan käsityksen siitä, mihin SAP Build Coden ja Joulen voisi odottaa pystyvän.

Sovelluskehitysprojektin suunnittelu aloitetaan kartoittamalla tapoja, joilla Joule-tekoälyä voidaan testata. Kartoituksen yhteydessä rajataan myös pois aiheet, joita opinnäytetyössä ei tulla käsittelemään, kuten tietoturvaseikat ja integroiminen SAPin tai kolmannen osapuolen palveluihin. Joulen arviointikriteeristöksi määritetään esimerkiksi Joulen kyky muuntaa erilaiset kehotteet koodiksi, SAPUI5-käytäntöjen ja oikean syntaksin noudattaminen,

sovelluslogiikan ylläpitäminen sekä sovelluksessa käytettävien tietojen sitomisen, mallin määrittelyn ja yleisen rakenteen tarkkuus.

Sovelluskehitysprojekti toteutetaan vaiheittain keskittyen niihin tehtäviin, joista Joulen uskotaan olevan apua. Joulelle annetut kehotteet ja sen luomat vastauksen kirjataan ylös muistiinpanoina, jotka tullaan purkamaan opinnäytetyöhön niiltä osin, kuin on tarpeellista työn loogisen seurattavuuden kannalta. Työvaiheita koskevien muistiinpanojen lisäksi kirjataan ylös yleisiä huomioita Joulen kanssa työskentelystä.

5.3 Tavoite ja tarkoitus

Työn lopputuloksena on tarkoitus selvittää, onko SAP Build Codea ja Joulea kannattavaa lähteä ottamaan osaksi toimeksiantajayrityksen sovelluskehitysprosesseja, vai onko sen hyvä antaa vielä kypsyä ja kehittyä lopulliseen muotoonsa.

Koska tekoäly on viime vuosina nostanut suosiotaan, halutaan opinnäytetyössä muodostaa kattava kuva siitä, mikä on Joulen potentiaali sovelluskehityksen työkaluna. Työssä otetaan käytännön sovelluskehitysprojektin avulla selvää, minkälaisissa kehitystehtävissä Joulea pystyttäisiin hyödyntämään ja mitkä ovat sen vahvuudet ja heikkoudet. Samalla tutkitaan, kuinka Joule ottaa huomioon kehitettävän projektin kokonaisuudessaan, kuinka se noudattaa sovelluskehityksen periaatteita ja SAPUI5:n käytäntöjä ja mitkä ovat sen vaikutukset koko sovelluskehitysprosessin sujuvuuteen ja tehokkuuteen.

6 SAP Build Code ja Joule käytännössä

SAP Build Coden ja Joulen käyttämistä sovelluskehityksen työkaluna testataan sovelluskehitysprojektin avulla. Projekti aloitetaan luomalla uusi SAP Fiori Application -sovellus SAP Build -alustan kautta. SAP Build Code tekee projektin alustuksen kehittäjälle helpoksi ja ohjaa käyttäjää prosessin läpi visuaalisen käyttöliittymän avulla.

Koska opinnäytetyön sovellusprojektin pohjalla käytetään toimeksiantajayrityksen projektimallia, suoritetaan sovelluksen alustus loppuun tuomalla työtilaan tyhjä projektimalli. Pohjan tuomisen jälkeen projektiin luodaan tiedostot tarvittaville näkymille, ohjaimille sekä malleille. Sovelluksen rakennetta pohjustetaan luomalla näkymätiedostoihin tarvittavat peruselementit ilman tarkempia parametrimäärittelyitä. Myös ohjain- ja mallitiedostoihin luodaan haluttu perusrakenne. Lisäksi kehitystyötä varten alustetaan testipalvelin (mockserver) ja luodaan tarvittaville tietotyypeille testidata.

Lopullisessa sovelluksessa esitetään taulumuotoista dataa. Käyttäjän on mahdollista vaihtaa taulussa näytettävää tietojoukkoa, hallinnoida aktiivisen taulun tietoja (lisäys, poisto) ja tallentaa tai perua tekemänsä muutokset. Sovelluksessa halutaan käyttää dynaamista taulunluontia ja modulaarista rakennetta.

6.1 XML-näkymän muokkaaminen

Sovelluksen käyttöliittymä on alustettu luomalla siinä käytettävälle taulukolle DataTable.fragment.xml-niminen tiedosto. Tiedostoon on lisätty XML-koodit sap.ui.table.Table-ohjainelementille sekä sen alapalkkiin sijoitetuille Peruuta- ja Tallenna-painikkeille (Cancel ja Save).

6.1.1 Pudotusvalikon lisääminen työkalupalkkiin

Sovelluksen kehitystyö aloitetaan luomalla taulukkoon pudotusvalikko, josta käyttäjä pystyy vaihtamaan aktiivista tietojoukkoa. Taulukkoelementti ei sisällä sisäänrakennettua metodia tietojoukon vaihtamiseksi. Jotta pudotusvalikko saadaan sijoitettua taulukon yläreunaan, tarvitaan sille extension-koosteen sisälle sijoitettava OverflowToolbar-ohjainelementti.

Pudotusvalikossa näytettävien taulujen nimet tallennetaan Data-malliin Tables-kokoelman alle avain-arvo-pareina.

Käyttäjätarinamuotoista kehotteella ei Joulen onnistu luoda käyttökelpoista koodia, vaan se pyrkii muuttamaan käytetyn taulukon sap.m.Table-ohjainelementiksi. Kun kehotetta tarkennetaan ja erikseen pyydetään käyttämään olemassa olevaa sap.ui.table.Table-ohjainelementtiä, jättää Joule taulukon entiselleen. Vastauksessaan se kuitenkin käyttää headerToolbar-koostetta, joka on vain sap.m.Table-ohjainelementin käytettävissä (Ohjelmakoodi 2).

Ohjelmakoodi 2 Väärän taulukko-ohjainelementin kooste

```
The "headerToolbar" name is neither a class name in the "sap.m"
namespace nor an aggregation of the "sap.ui.table.Table"
class. UI5 Language Assistant
View Problem (Alt+F8)    No quick fixes available
<headerToolbar>
    <Toolbar>
        <content>
        <ComboBox items="{Data>/Tables}" selectionChange="onTableChange">
            <items>
            <core:Item key="{Data>key}" text="{Data>name}"/>
            </items>
        </ComboBox>
        </content>
    </Toolbar>
</headerToolbar>
```

Joulelle annettavaa kehotetta muutetaan suoraan ohjeistuksen muotoon, mutta saatu vastaus pysyy samana. Vasta Joulen kanssa käydyn keskustelun tyhjentäminen saa sen muuttamaan vastaustaan. Vaikka suoran ohjeistuksen mukaan annetun kehotteen mukaan Joule osaa sijoittaa luomansa koodin oikein, jättää se pudotusvalikon kokonaan huomiotta ja korvaa sen valikkopainikkeella (Ohjelmakoodi 3).

Ohjelmakoodi 3 Virheellinen valikko-ohjainelementti

```
<t:extension>
    <OverflowToolbarButton
        icon="sap-icon://menu2"
        press="onMenuPress"
    />
</t:extension>
```

Keskustelutapaa Joulen kanssa muutetaan ja seuraavaa kehotetta ei kirjoiteta suoraan keskustelunäkymään. Sen sijasta halutut elementit ja niiden toiminnot kirjoitetaan kommentteina tiedostoon siihen kohtaan, johon ne halutaan luotavan (Ohjelmakoodi 4). Joulelle annettaan kehotteeksi jatkaa DataTable-näkymää siihen lisättyjen kommenttien mukaisesti.

Ohjelmakoodi 4 Tiedostoon lisätyt ohjeet

```
<t:extension>
    // OverflowToolbar, with ToolbarSpacer on right
    // Label with i18n text 'Select Table', use same naming convention as in other i18n keys
    // Select component with items of property /Tables of model Tables. SelectedKey should
    refer to property /SelectedTable in model Tables
</t:extension>
```

Tarkkojen ja suoraan tiedostoon lisättyjen kommenttien pohjalta Joule generoi vastauksen, jota pystytään lähes sellaisenaan hyödyntämään. Se luo halutut ohjainelementit, osaa huomioida muut i18n-resurssimalliin luodut avaimet ja noudattaa niissä käytettyä nimeämiskäytäntöä. Joulen vastausta joudutaan kuitenkin vielä muuttamaan tietojen sitomisen syntaksin osalta.

XML-tiedoston alussa siihen tuodaan käytettävissä elementeissä tarvittavia moduuleita. Moduuleille määritetään etuliitteet, joita käytetään myöhemmin osoittamaan kirjastoa, josta ohjainelementti sijaitsee. DataTable.fragment.xml-tiedostoon on tuotu sap.ui.core etuliitteellä ”c” ja sap.ui.table etuliitteellä ”t” (Ohjelmakoodi 5).

Ohjelmakoodi 5 Taulukkotiedostoon tuodut moduulit

```
<c:FragmentDefinition
    xmlns="sap.m"
    xmlns:c="sap.ui.core"
    xmlns:t="sap.ui.table"
>
    <t:Table···
    >
        <t:extension>···
        </t:extension>

        <t:footer>···
        </t:footer>
    </t:Table>
</c:FragmentDefinition>
```

Joule ei kuitenkaan huomioi tiedostoon tuotuja moduuleita oikein, vaan pyrkii käyttämään pudotusvalikon sisällä olevan Item-ohjainelementin etuliitteenä ”core” tiedostossa määritetyn ”c”:n sijasta. Lisäksi Joulen käyttämät avaimet (”key” ja ”text”) eivät vastaa Data-malliin määritettyjä avaimia. (Ohjelmakoodi 6)

Ohjelmakoodi 6 Virheellinen etuliite ja avaimet

```
<t:extension>
    <OverflowToolbar>
        <ToolbarSpacer />
        <Label text="{i18n>SELECT_TABLE}" />
        <Select
            items="{Tables>/Tables}"
            selectedKey="{Tables>/SelectedTable}"
        >
            <core:Item key="{Tables>key}" text="{Tables>text}" />
        </Select>
    </OverflowToolbar>
</t:extension>
```

Määrittämättömään etuliitteeseen viittaaminen aiheuttaa sovelluksen käynnistäessä suoritusvirheen, joka estää sovelluksen käyttämisen. Vastauksessa generoitu koodi saadaan käyttökelpoiseksi vaihtamalla tietojen avaimet vastaamaan mallissa määritettyjä (”Key” ja ”Description”) ja vaihtamalla Item-ohjainelementin etuliite ”c”:ksi (Ohjelmakoodi 7).

Ohjelmakoodi 7 Korjattu ohjainelementtikoodi

```
<t:extension>
    <OverflowToolbar>
        <ToolbarSpacer />
        <Label text="{i18n>SELECT_TABLE}" />
        <Select
            items="{Tables>/Tables}"
            selectedKey="{Tables>/SelectedTable}"
        >
            <c:Item key="{Tables>Key}" text="{Tables>Description}" />
        </Select>
    </OverflowToolbar>
</t:extension>
```

6.1.2 Taulukon rivitoiminnon muuttaminen

Sovelluksessa käytettävällä sap.ui.table.Table-ohjainelementin rivien oletusarvoinen toiminto on navigointi. Käyttäjän painaessa riviä, voidaan hänet ohjata toiseen näkymään esimerkiksi tarkastelemaan rivillä esitetyn objektin laajempia tietoja. Koska kehitettävässä sovelluksessa käsiteltävät tietojoukot ovat yksinkertaisia ja ne pystytään näyttämään kokonaisuudessaan taulukon riveillä, muutetaan myös rivitoiminto. Käyttäjälle halutaan antaa mahdollisuus poistaa rivejä, mikä pystytään toteuttamaan jokaisen rivin oikeaan laitaan näkyviin tulevan X-ikonia painamalla.

Sovelluksen sap.ui.table.Table-ohjainelementillä ei ole valmista parametriä, jonka kautta rivitoimintoa voidaan muuttaa. Poistotoimintoa varten tulee taulukon XML-koodiin luoda rowActionTemplate-kooste, jonka sisälle sijoitetaan RowAction- ja RowActionItem-ohjainelementit.

Tehtävä annetaan Joulelle hyvin yksinkertaisen kehotteen avulla: muuta rivitoiminto poistoksi (”change row action to Delete”). Vaikka rivitoiminnon muuttaminen ei ole monimutkainen toteutus, luo Joule ensimmäisellä yrittämällä vastauksen, jonka jo SAP Build Application Studion koodieditori merkkaa virheelliseksi. Vastauksessaan Joule asettaa suoraan taulukon parametreihin uuden rowActionTemplate-avaimen, jota taulukko ei tunnista lainkaan (Ohjelmakoodi 8).

Ohjelmakoodi 8 Virheellinen rowActionTemplate-parametri

```
busyIndicatorDelay= 0
rowActionTemplate="{
    path: 'Data>/Items',
    templateShareable: false,
    template: {
        Type: 't:RowAction',
        items: {
            path: 'Data>/Items',
            templateShareable: false,
            template: {
                Type: 't:RowActionItem',
                type: 'Delete',
                press: '.onDeletePress'
            }
        }
    }
}"
```

Virheilmoitus annetaan Joulen käsiteltäväksi, jonka jälkeen se osaakin korjata koodin ja tuottaa vastaukseksi toimivan version (Ohjelmakoodi 9). Kun taulukkoon tuodaan myöhemmin tietojoukkoja, saadaan rivien loppuihin näkymään myös poistopainikkeet. Rivitoiminnot piilotetaan toistaiseksi ja ne tullaan ottamaan takaisin käyttöön myöhemmin tietojen muokkaustoiminnallisuuksien lisäämisen yhteydessä.

Ohjelmakoodi 9 Toimiva rivitoimintokoodi

```
<t:rowActionTemplate>
    <t:RowAction>
        <t:items>
            <t:RowActionItem type="Delete" press=".onDeletePress" />
        </t:items>
    </t:RowAction>
</t:rowActionTemplate>
```

6.2 Listasidonta tehdasfunktiolla

SAPUI5:n listasidontaa ja tehdasfunktiota hyödyntämällä sovelluksen tietojoukot pystytään esittämään käyttäjälle ilman erillisiä XML-tiedostoja ja kovakoodattuja sarakkeita ja rivejä.

Projektin Tables-malliin luodaan Approvers-tietojoukolle yksinkertainen konfiguraatio-objekti sekä erillinen funktio varsinaisten sarakekonfiguraatioiden luomiseksi mallin Tables-parametriin.

Konfigurointifunktiolla muotoillaan sille välitetty sarakekonfiguraatio UI:ssa käytettävien sarakkeiden hyväksymiksi parametreiksi (Ohjelmakoodi 10). Kunkin parametrin avainta käytetään sarakkeen nimen hakemisessa i18n-resurssimallista. Muiden parametrien arvot haetaan konfiguraatio-objektista ja niille määritellään vaihtoehtoinen arvo sen varalta, ettei konfiguraatio-objektiin ole määritetty vastaavia avaimia.

Ohjelmakoodi 10 Konfiguraatiofunktio ja -objekti

```
34      generateColumnConfiguration: function() {
35          const oConfig = this._approverConfig();
36          const aKeys = Object.keys(oConfig || {});
37          return aKeys.map((sKey) => {
38              const oColumn = oConfig[sKey] || {};
39              return {
40                  Label: `COLUMN_LABEL_${sKey.toUpperCase()}`,
41                  Type: oColumn.Type || 'Input',
42                  Property: oColumn.Property || sKey,
43                  Required: oColumn.Required || false
44              };
45          });
46      },
```

```
49      _approverConfig: function() {
50          return {
51              Plant: {},
52              MaterialGroup: {},
53              ResponsiblePerson: {
54                  Required: true
55              },
56              ApprovalRuleId: {
57                  Required: true
58              }
59          };
60      },
61
```

Jotta malliin tallennetut sarakekonfiguraatiot saadaan näkymään taulukossa, tulee taulukkoa käsittelevään Main-ohjaimeen luoda tehdasfunktio. Myöhemmin taulukolle lisätään sarakeparametriksi yhteys mallin konfiguraatioon sekä Main-ohjaimeen luotuun tehdasfunktioon, joiden avulla sarakkeet generoidaan dynaamisesti.

Vaikka tehdasfunktio on SAPUI5:n sisäänrakennettu standardimetodi, vaatii hyväksyttävän vastauksen saaminen useampaa kehotetta. Siitä huolimatta, että kehotteessa erikseen pyydetään käyttämään tehdasfunktiota, ei Joule osaa hyödyntää niitä vastauksissaan (Ohjelmakoodi 11). Vaikka Joulen luomalla koodilla voitaisiinkin saada sarakkeet taulukkoon, ei se vastaa kehotetta eikä näin ollen ole hyväksyttävä.

Ohjelmakoodi 11 Tehdasfunktiota vastaamaton sarakkeenluontifunktio

```
createColumns: function(oTable, oModel) {
    var aColumns = oModel.getProperty('/Columns');
    aColumns.forEach(function(oColumn) {
        var oTableColumn = this._createColumn(oColumn);
        oTable.addColumn(oTableColumn);
    }.bind(this));
},
```

Keskustelun tyhjentämisen ja kahden tarkennuksen jälkeen Joulen onnistuu luoda SAPUI5-dokumentaatiota vastaava tehdasfunktio sarakkeiden luomiselle. Luotuja koodeja joudutaan vielä hienosäätämään, sillä kaikki funktiossa käytetyt parametrien avaimet eivät vastaa malliin tallennettuja. Lisäksi Joule lisää sarakkeiden parametreihin arvoja, joita ei ole mallissa määritetty lainkaan. Näiden poistamisen lisäksi solujen syöte-elementtien luomiseen käytetty if-lauseke pyydetään vaihtamaan switch-lausekkeeseen.

Joule ei tälläkään kertaa noudata kehotetta kovinkaan tarkasti ja poistamisen sijasta se siirtää mallista löytymättömät parametrit omien if-lausekkeidensa sisälle. Sarakekonfiguraatiossa määritetyillä vaihtoehtoisilla arvoilla pyritään välttämään juuri Joulen vastauksen mukaista koodia. Myös Joulen switch-lausekkeessa esiintyy ohjelmistokehityksen ”älä toista itseäsi”-periaatteen vastaista koodia, sillä Input-tapaus esiintyy kahdesti. Mallissa määritettyä required-parametriä ei ole huomioitu ja toisaalta editable-parametri on asetettu turhaan, sillä sen vakioarvo on tosi. (Ohjelmakoodi 12)

Ohjelmakoodi 12 Joulen luoma switch-lauseke

```
createColumn: function(sId, oContext) {
    var oColumn = new sap.ui.table.Column(sId, {
        label: this.createLabel(oContext.getProperty('Label')),
        template: this.createTemplate(oContext.getProperty('type'), oContext.getProperty('path'))
    });
    if (oContext.getProperty('sortProperty')) {
        oColumn.setSortProperty(oContext.getProperty('sortProperty'));
    }
    if (oContext.getProperty('filterProperty')) {
        oColumn.setFilterProperty(oContext.getProperty('filterProperty'));
    }
    if (oContext.getProperty('width')) {
        oColumn.setWidth(oContext.getProperty('width'));
    }
    return oColumn;
},

createTemplate: function(sType, sPath) {
    switch (sType) {
        case 'Text':
            return new sap.m.Text({
                text: '{' + sPath + '}'
            });
        case 'Input':
            return new sap.m.Input({
                value: '{' + sPath + '}',
                editable: true
            });
        // add cases for other input types as needed
        default:
            return new sap.m.Input({
                value: '{' + sPath + '}',
                editable: true
            });
    }
},
```

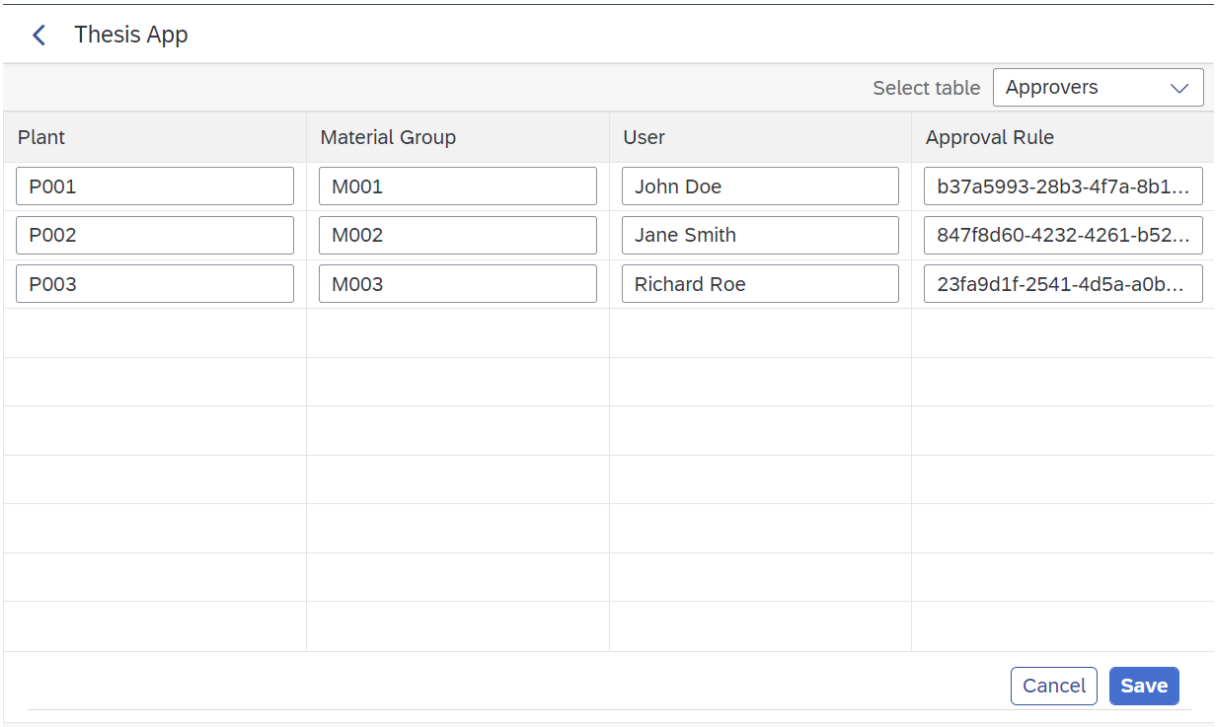
Koodi korjataan käsin poistamalla ylimääräiset if-lausekkeet, asettamalla syöte-elementin nimi (label) osoittamaan i18n-resurssimallin avainta ja poistamalla ylimääräinen Input-tapaus switch-lausekkeesta. Jotta createTemplate-funktiossa voidaan käsitellä kokonaisia objekteja, muutetaan funktio vastaanottamaan createColumn-funktiossa haettu oColumn-objekti yksittäisten parametrien sijasta. Viimeiseksi Joulea pyydetään sisällyttämään mallin nimi tietojen sitomisen syntaksiin. Lopuksi XML-taulukkoelementtiin lisätään tarvittavat sarake- ja riviparametrit (columns ja rows) (Ohjelmakoodi 13).

Ohjelmakoodi 13 Taulukko-ohjainelementin parametrit

```
<t:Table
    id="dataTable"
    columns="{
        path: 'Tables>/Columns',
        factory: '.columnsFactory'
    }"
    rows="{
        path: 'Data>/Items'
    }"
    selectionMode="None"
    rowActionCount="1"
    busy="{Data>/IsBusy}"
    busyIndicatorDelay="0"
>
```

Tehtyjen muutosten jälkeen testipalvelimella käynnistetyssä sovelluksessa voidaan tarkastella Approvers-tietojoukon tietoja taulukkomuodossa. Sovelluksessa näkyy myös sinne aikaisemmin rakennettu pudotusvalikko sekä sovelluksen alustusvaiheessa lisätyt Cancel- ja Save-painikkeet (Peruuta ja Tallenna). (Kuva 2)

Kuva 2 Sovelluksen taulukkonäkymä



6.3 Funktioiden luominen ja jatkaminen

Koska aiemmin luotu generateColumnConfiguration-funktio ei ole tietoinen pudotusvalikon kautta valitusta taulusta, näyttää sovellus toistaiseksi ainoastaan Approvers-tietojoukon. Jotta tietojoukko saadaan päivittymään valitun taulun mukaisesti, tulee funktion vastaanottaa valittu taulu parametrinä. Lisäksi ApprovalRules-tietojoukolle luodaan oma konfiguraatio-objektinsa, jonka mukaan sovellus osaa muuttaa taulukon sarakkeet vastaamaan tietojoukon rakennetta.

Joulea kehotetaan luomaan konfiguraatio-objekti, jonka rakenne noudattaa aiemmin luotua _approverConfig-objektia. Joule ei kuitenkaan osaa hakea testidata-kansioon tallennettua rakennetta tietojoukon nimen perusteella, eikä sen käyttämät objektin avaimet vastaa tietojoukon todellista rakennetta. Kehotetta korjataan liittämällä haluttu tietorakenne sen mukaan ja pyytämällä Joulea luomaan objekti uudelleen sitä noudattaen. Korjatun kehotteen jälkeen vastaukseksi saadaan lähes halutunlainen objekti (Ohjelmakoodi 14).

Ohjelmakoodi 14 Joulen luoma konfiguraatio-objekti

```
_approvalRuleConfig: function() {
    return {
        Id: {
            Type: 'Edm.Guid',
            Nullable: false
        },
        Description: {
            Type: 'Edm.String',
            MaxLength: 100
        },
        Level: {
            Type: 'Edm.Int32'
        },
        Areas: {
            Type: 'Collection(Edm.String)',
            Nullable: false
        }
    };
},
```

Joulen vastaus on sinällään käyttökelpoinen. Jotta konfiguraatio-objekti olisi kuitenkin yhdenmukainen aiemmin luodun _approverConfig-objektin kanssa, halutaan siihen tehtävän pieniä korjauksia. Joulea pyydetään toteuttamaan seuraavat muutokset:

1. korvaa Nullable avaimella Required ja sisällytä parametri vain, jos arvo ei voi olla tyhjä

2. muuta Edm.String arvoksi ”Input” ja Edm.Int32 arvoksi ”Number”
3. muuta kaikki Collection-tyypit ohjainelementeiksi, jotka tukevat useita syötteitä

Joule noudattaa kehotteen kahta viimeistä kohtaa hyvin ja osaa myös valita Collection-tyypeille oikean MultiInput-ohjainelementin. Nullable-parametrien muuttamisen se kuitenkin tulkitsee väärin ja asettaa vastauksessaan kaikille parametreille ”Required: true”. Kehotetta joudutaan muuttamaan kaksi kertaa, ja lopulta haluttu vastaus saadaan hyvin pseudokoodimaisella kehotuksella: jos Nullable: epätosi, korvaa se Required: tosi (”if Nullable: false, replace it with Required: true”).

Kun annettu kehote on riittävän yksinkertainen, suoriutuu Joule valmiiden funktioiden toiminnallisuuksien jatkamisesta hyvin. Joulea pyydetään luomaan funktio, joka palauttaa saamaansa parametriä vastaavan taulukkokonfiguraation sekä muuttamaan generateColumnConfiguration hyödyntämään tätä palautettua arvoa oikeiden sarakekonfiguraatioiden luomisessa. Kun Joulelle annetaan kehote syöte-elementtien luomisesta vastaavan funktion jatkamisesta, joudutaan koodia työstämään taas useamman kerran.

Kun kehotteessa pyydetään switch-lausekkeen jatkamista käsittelemään Number- ja MultiInput-tyypit, tuottaa Joule niille omat tapauksensa, mutta jättää elementin varsinaisen sisällön luomatta. Teknisesti ottaen Joulen vastaus täyttää kehotteen minimivaatimukset. Koska Joulen on kuitenkin mainostettu pystyvän huomioimaan tiedostoissa käytetyt koodikokonaisuudet ja switch-lausekkeen muissa tapauksissa esimerkiksi path-parametri on täytetty, voisi vastaus olla viimeistellympi (Ohjelmakoodi 15).

Ohjelmakoodi 15 Joulen lisäämät tyyppitapaukset

```
case "Number":
    return new sap.m.Input({
        value: {
            path: '',
            type: 'sap.ui.model.type.Float'
        }
    });
case "MultiInput":
    return new sap.m.MultiInput({
        tokens: {
            path: ''
        }
    });
```

Koodi ei myöskään ole sellaisenaan käyttökelpoinen. SAPUI5:n MultiInput-ohjainelementti on syöte, johon käyttäjä voi valita useita itsenäisiä poletteja (token). Kun ohjainelementtiä käytetään suoraan XML-näkymään kirjoitettuna, sijoitetaan Token-ohjainelementit tokens-koosteen sisälle (Ohjelmakoodi 16).

Ohjelmakoodi 16 MultiInput XML-toteutuksena

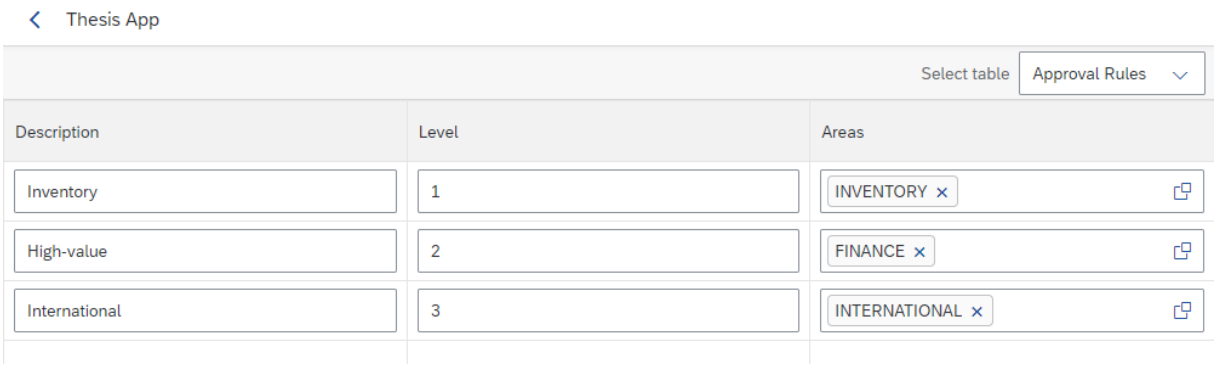
```
<MultiInput
    id="multiInput"
    tokens="{
        path: '/Tokens'
    }"
>
    <tokens>
        <Token
            key="{Key}"
            text="{Description}"
        />
    </tokens>
</MultiInput>
```

Koska Joulen tuottamassa koodissa ei luoda Tokens-ohjainelementtejä lainkaan, aiheuttaa koodin ajaminen sellaisenaan virheen: ”Malli tai tehdasfunktio tokeneita varten puuttuu” (”Missing template or factory function for aggregation tokens”). Virheilmoitus syötetään Joulelle sellaisenaan ja vastauksena saadaan virheelle selite sekä korjattu koodi. Vastauksessa luodaan nyt Tokenit hyödyntäen niille omaa, taulukossakin aiemmin hyödynnettyä tehdasfunktiota eikä sovelluksen käynnistäminen aiheuta enää virheitä. Joulen vastausta joudutaan kuitenkin tälläkin kertaa korjailemaan, sillä sen ei onnistu käyttämään tietojen sitomisessa oikeaoppista syntaksia. Kun arvojen syntaksiin lisätään viittaus käytettyyn malliin (Ohjelmakoodi 17), saadaan Areas-arvoihin valitut poletit näkymään halutusti käyttöliittymän taulukossa (Kuva 3).

Ohjelmakoodi 17 Korjattu MultiInput-ohjainelementin luontikoodi

```
case 'MultiInput':
    var oMultiInput = new sap.m.MultiInput();
    oMultiInput.bindAggregation('tokens', {
        path: 'Data>' + oContext.getProperty('Property'),
        factory: function(sId, oContextItem) {
            return new sap.m.Token({
                key: '{Data>' + oContextItem + '}',
                text: '{Data>' + oContextItem + '}'
            });
        }
    });
```

Kuva 3 Taulukon MultiInput-syötteessä näkyvät poletit



6.4 Apudialogin liittäminen syöteohjainelementteihin

Taulukon Areas-sarakkeen syötteiden oikeassa reunassa on nähtävissä apudialogi-ikoni (Value Help). Ikoni ilmaisee käyttäjälle, että kyseiseen kenttään on saatavilla lista hyväksyttäviä arvoja, jotka voidaan ikonia painamalla avata listamuotoisena dialogina. Itse dialogi poimii sen listaamat arvot dialogiohjainelementille välitetyn mallin ja polun avulla.

6.4.1 Aputietojen hakeminen

Jotta sovellukseen saadaan käyttöön projektitiedostoon tallennetut testidatat, lisätään myös Approvers-taulunäkymän Plant-, Material Group- ja Approval Rule -sarakkeille apudialogi-ikonit. Koska nämä syötteet hyväksyvät vain yhden arvon, ei itse syöteohjainelementtiä muuteta MultiInput-ohjainelementiksi.

Jotta apudialogi saadaan lisättyä dynaamisesti halutuille sarakkeille, täytyy Approvers-taulun konfiguraatio-objektia laajentaa sisältämään apudialogin tarvetta sisältävä parametri. Kun kehote annetaan Joulelle, ei se osaa kohdistaa luomiaan muutoksia olemassa olevaan malliin ja objektiin, vaan se pyrkii tekemään muutoksia projektipohjassa olemassa olevaan, mutta sovelluksessa käyttämättömään kustomoituun ohjainelementtiin. Kehotteen säätö eikä keskustelun nollaus muuta Joulen vastausta, mutta se saadaan lopulta tuottamaan halutunlainen vastaus tiedostoon lisättyjen kommenttien avulla. Muutettavan funktion yläpuolelle sijoitetaan kommenttimuotoinen kehote, jota Joule noudattaa uudessa vastauksessaan (Ohjelmakoodi 18). Muutokset tehdään vain Approvers-konfiguraatiolle, sillä Approval Rules -tietojoukon Areas-kentässä käytettävässä MultiInput-ohjainelementissä aputieto on oletuksena päällä.

Ohjelmakoodi 18 Aputieto-parametrit

```
44
45      generateColumnConfiguration: function(sTableKey) {
46          const oConfig = this.generateConfigForTable(sTableKey);
47          const aKeys = Object.keys(oConfig || {});
48          return aKeys.map((sKey) => {
49              const oColumn = oConfig[sKey] || {};
50              return {
51                  Label: `COLUMN_LABEL_${sKey.toUpperCase()}`,
52                  Type: oColumn.Type || 'Input',
53                  Property: oColumn.Property || sKey,
54                  Required: oColumn.Required || false,
55                  ShowValueHelp: oColumn.ShowValueHelp || false // Added ShowValueHelp property
56              };
57          });
58      },
59
60      // Extend object to include value help icons for Plant, MaterialGroup and ApprovalRuleId
61      _approverConfig: function() {
62          return {
63              Plant: {
64                  ShowValueHelp: true
65              },
66              MaterialGroup: {
67                  ShowValueHelp: true
68              },
69              ResponsiblePerson: {
70                  Required: true
71              },
72              ApprovalRuleId: {
73                  Required: true,
74                  ShowValueHelp: true
75              }
76          };
77      },
78
```

Kun solujen sisällön generoivan funktion Input-tapaukseen nyt välitetään showValueHelp-parametri, voidaan sovelluksessa nähdä aputietoikonit myös Approvers-taulukon kentissä. Ikonit eivät avaa apudialogia automaattisesti, vaan niihin tulee liittää valueHelpRequest-tapahtuman käsittelevä funktio, joka huolehtii parametrien välittämisestä avattavalle apudialogille. Apudialogina käytetään projektipohjaan luotua, SAPUI5:n standardiapudialogista yksinkertaistetumpaa versiota.

Koska projektipohjan dialogin tarvitsemat parametrit ovat kattavasti dokumentoitu JSDoc:illa, pyydetään Joulea luomaan tapahtumankäsittelyfunktio, joka avaa dialogin ja välittää sille dokumentaatiossa määritetyt pakolliset parametrit. Vaikka Joule osaa vastauksessaan hyödyntää toivottuja metodeita ja oikeaa dialogiohjainelementtiä, ei sen dialogille välittämiä parametrejä ole määritetty mihinkään. Vastausta joudutaan korjailemaan pyytämällä vielä erikseen vain JSDoc:issa määritettyjen parametrien käyttämistä ja ainoastaan pakollisten parametrien sisällyttämistä dialogin avaamiseen. Korjauksilla funktioon saadaan oikeat parametrin, mutta Joule jättää silti mukaan myös muutamia vapaaehtoisia parametrejä, jotka päädytään poistamaan dialogin avauskutsusta käsin. (Ohjelmakoodi 19)

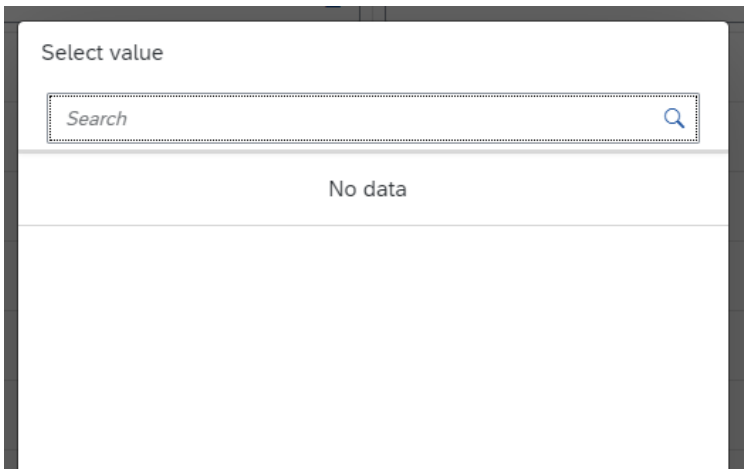
Ohjelmakoodi 19 Valmis valueHelpRequest-tapahtuman käsittelevä funktio Plants-tietojoukolle

```
onPlantValueHelpRequest: function() {
    this.openDialog('SearchDialog', {
        configurationModelName: 'Helpers',
        pathToModel: '/Plants',
        listItemValueKey: 'Plant',
        listItemDescriptionKey: 'PlantDescription',
        saveModelName: 'Data',
        firstValueToSave: 'DisplayItems/Plant'
    });
},
```

Hyväksytyn funktion luomisen jälkeen Joulea pyydetään luomaan vastaavat funktiot kaikille tarvittaville aputietojoukoille. Joule suoriutuu kehotteesta hyvin. Koska kehitettävässä sovelluksessa on vain muutama aputietodialogia hyödyntävä syöte, ei tapahtumankäsittelyfunktioita optimoida pidemmälle. Mikäli sovelluksessa olisi useita apudialogisyötteitä, olisi koodi ylläpidettävämpää ja ”älä toista itseäsi”-periaatteen mukaisempaa, mikäli tapahtumafunktiot ottaisivat parametreinä dialogin avaamiseen tarvittavat tiedot. Tällöin parametrien arvot voitaisiin määrittää esimerkiksi taulukonfiguraatio-objektissa.

Joulea pyydetään liittämään luotuihin syöte-elementteihin valueHelpRequest-tapahtuma, jolla määritetään painiketapahtumasta kutsuttava funktio. Joule lisää aluksi tapahtuman vain Input-ohjainelementille, mutta luo tarkennuksen jälkeen toimivan ratkaisun myös MultiInput-ohjainelementille. Sovelluksesta voidaan nyt mitä tahansa aputietoikonia painamalla avata apudialogi. Koska dialogeissa tarvittavaa dataa ei ole vielä ladattu mihinkään malliin sovelluksen käytettäväksi, saadaan toistaiseksi näkyviin tyhjä dialogi. (Kuva 4)

Kuva 4 Tyhjä apudialogi



6.4.2 Aputietojen hakufunktion jatkaminen

Jotta sovelluksen Approval Rule -sarakkeelle saadaan haettua lista kaikista Approval Rules -tietojoukon riveistä, pitää aputiedot hakevaa funktiota laajentaa. Haluttua hakuosoitetta ei voida suoraan lisätä funktion endpoints-muuttujaan, sillä tiedot tullaan hakemaan toisesta palvelusta.

Joulelle annetaan kehote, jossa pyydetään ApprovalRules-tietojen hakemista ADMIN_SRV-palvelusta. Vaikka sen vastauksen koodi toimii ja sillä saadaan tarvittavat tiedot sovelluksen käytettäviksi, vaatii koodi korjauksia. Koska Joule tallentaa palvelut ja hakupolut omiin taulukkomuuttujiinsa, ei kutsuja voida helposti luoda yhden iteratiivisen metodin sisällä (Ohjelmakoodi 20). Joule luokin molemmille palveluille omat, täysin identtiset forEach-metodit.

Ohjelmakoodi 20 Taulukkomuuttujat

```
var helperServiceName = 'HELPER_SRV';
var adminServiceName = 'ADMIN_SRV';
var helperEndpoints = ['ApprovalAreas', 'MaterialGroups', 'Plants'];
var adminEndpoints = ['ApprovalRules'];
```

Joulea pyydetään muuntamaan palvelut ja hakupolut yhdeksi objektiksi, jota voidaan käyttää forEach-metodissa. Vastaukseksi saadaankin tällä kertaa siistitty koodi, joka on helppolukuista eikä toista itseään. Joule muuttaa erilliset taulukkomuuttujat objektiksi, joka käsitellään yksittäisen forEach-metodin sisällä. (Ohjelmakoodi 21). Ylimääräisen forEach-metodin Joule poistaa koodista kokonaan.

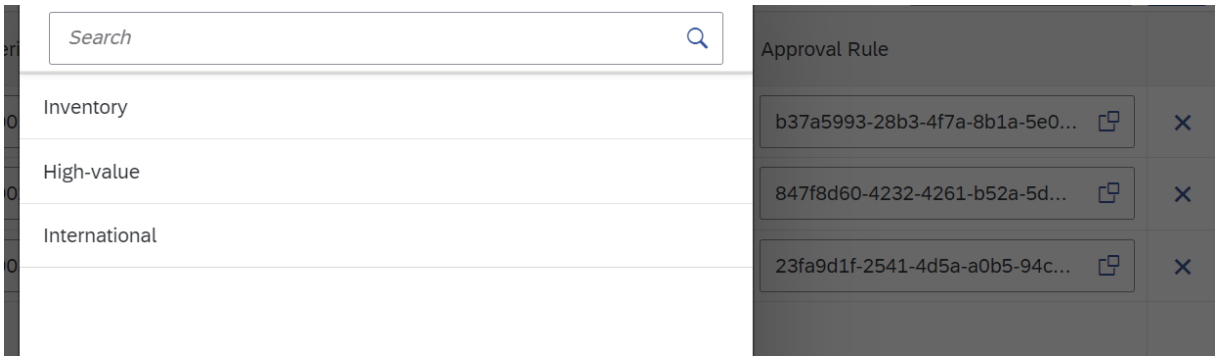
Ohjelmakoodi 21 Iteroitavissa oleva muuttuja

```
var services = [
    {
        serviceName: 'HELPER_SRV',
        endpoints: ['ApprovalAreas', 'MaterialGroups', 'Plants']
    },
    {
        serviceName: 'ADMIN_SRV',
        endpoints: ['ApprovalRules']
    }
];
```

6.5 Arvojen muotoilu

Sääntöjen apudialogista käyttäjä saa nyt eteensä tekstikuvaukset kaikista Approval Rules -taulukkoon tallennetuista rivistä. Approver-tietojoukon Approval Rule taas säilyttää viittauksen asetetun säännön Id-arvoon, joten käyttökokemus on tallaisenaan hyvin sekava. (Kuva 5). Koska molemmista taulukoista on myös piilotettu rivien id-arvot, ei käyttäjän voida olettaa pystyvän yhdistämään näkemäänsä id:tä rivien kuvauksiin.

Kuva 5 Etualalla apudialogin arvot, taustalla syötteen arvot



Jotta Approval Rule -syötteessä pystytään säilyttämään viittaus säännön id-arvoon, mutta näyttää käyttäjälle vain tekstimuotoinen kuvaus, tulee ohjainelementtiin liittää muotoilufunktio. Funktion tulee ottaa vastaan argumentteina säännön id sekä kokoelma kaikista Helpers-malliin tallennetuista Approval Rules -objekteista. Näiden avulla voidaan etsiä taulukosta id:tä vastaava objekti ja palauttaa ainoastaan sen Description-arvo käyttäjän nähtäville. Kaikki sovelluksessa käytettävät muotoilufunktiot säilytetään util/formatter.js-tiedoston sisällä, josta niiden hallinnoiminen on helppoa.

Joulelle annetaan muotoilufunktion luomista varten yksityiskohtainen kehote: “extend _approverConfig/ApprovalRuleId to include properties named 'Property' and 'Formatter'. Property should be a string, which contains of key of the parent item and a reference to Helpers/ApprovalRules. Values must be separated by a comma. For Formatter, implement properly named formatter function, which takes values saved in Property as arguments and then returns ApprovalRuleDescription corresponding to received ApprovalRuleId.”

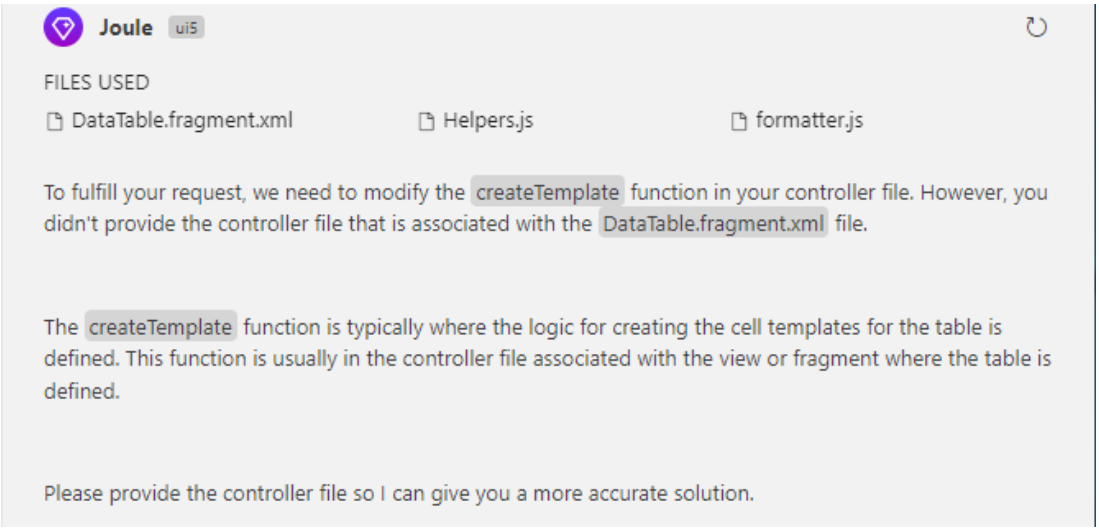
Kehotteen avulla saadaan Joulelta vastaukseksi funktio, joka täyttää kaikki kehotteessa toivotut seikat. Joule ei kuitenkaan automaattisesti huomaa asettaa funktiota formatter-tiedoston sisälle, vaan pyrkii lisäämään sen suoraan taulukonfiguraatio-objektin sisälle. (Ohjelmakoodi 22). Vastausta täydennetään vielä lisäämällä uudet parametrit sarakkeen konfiguraation palauttavaan funktioon. Myös Property-arvosta korjataan syntaksivirhe muuttamalla arvon viimeinen osa muotoon ”Helpers>/ApprovalRules”.

Ohjelmakoodi 22 Muotoilufunktio konfiguraatio-objektissa

```
showValueHelp: true,
Property: 'ApprovalRuleId,Helpers/ApprovalRules',
Formatter: function(sApprovalRuleId, aApprovalRules) {
    const oApprovalRule = aApprovalRules.find((oRule) => {
        return oRule.Id === sApprovalRuleId;
    });
    return oApprovalRule ? oApprovalRule.Description : '';
}
```

Jotta muotoiltu arvo saadaan vietyä käyttöliittymän syötteen ohjainelementtiin, tulee Main-ohjaimen tehdasfunktioon tehdä muutoksia. Näitä toiminnallisuutta lähdetään toteuttamaan yksi kehote kerrallaan, mutta heti ensimmäisen kohdalla törmätään työvaiheen keskeyttävään ongelmaan: kun Joulea pyydetään jatkamaan createTemplate-funktiota annettujen ohjeiden mukaisesti, saadaan vastaukseksi ilmoitus, ettei Joulella ole pääsyä koko Main-ohjaintiedostoon. Siitä huolimatta, että ohjaintiedosto löytyy projektin tiedostoista, se on SAPUI5:n käytäntöjen mukaan nimetty ja Joule on siihen aiemmin tehnyt muutoksia, on tiedosto kadonnut Joulen tietoisuudesta kokonaan.

Kuva 6 Vastaus, jossa kehotetta koskenutta tiedostoa Joule ei löydä



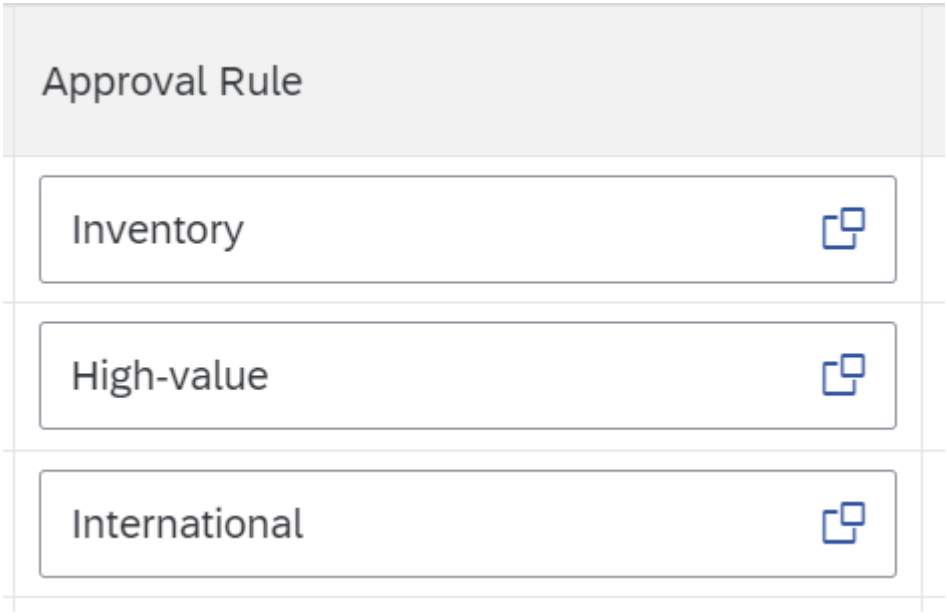
Funktioon tarvittavat muutokset päädytään lisäämään koodiin ilman Joulen avustusta. Funktion alkuun lisätään Property-arvon käsittelevä forEach-metodi ja jokaisen tyyppitapauksen arvo muutetaan muotoon, jossa viitataan parts-muuttujan sisällä oleviin polkuihin. Lisäksi Input-ohjainelementille määritetään formatter-parametri, joka viittaa aiemmin luotuun, formatter-tiedostoon siirrettyyn muotoilufunktioon (Ohjelmakoodi 23).

Ohjelmakoodi 23 Muotoiluparametrin sisältävä arvo

```
const oInput = new sap.m.Input({
    value: {
        parts: aParts,
        formatter: fnFormatter ? this.formatterUtil[fnFormatter] : this.formatterUtil.formatInputValue()
    },
    required: oContext.getProperty('Required')
```

Muutosten jälkeen sovelluksen Approval Rule -syötteissä nähdään nyt id-arvojen sijasta kuvaukset, jotka vastaavat apudialogissa näytettävää listausta (Kuva 7).

Kuva 7 Syötteiden muotoillut arvot



6.6 Asynkroniset rajapintakutsut

Apudialogeja varten projektin testidatakansioon on projektin alkuvaiheessa luotu testidatatiedostot Approval Areas -, Plant- ja Approval Rule -tietojoukoille. Approval Rule -tietojen apudialogin halutaan näyttävän lista kaikista Approval Rules -taulukon riveistä.

Käyttökokemuksen parantamiseksi syötteiden vaatimat aputiedot halutaan ladata ennen, kuin käyttäjä pääsee tekemään muutoksia taulukon tietoihin. Tämän saavuttamiseksi kaikki aputietojen lataamiseen tarvittavat rajapintakutsut tehdään JavaScriptin Promise.all-metodin sisällä. Taulukkoon lisätään varattu-ilmaisin (busyIndicator), jonka arvo liitetään mallissa hallinnoituun IsBusy-parametriin. Kun taulukon busyIndicator-parametrille välitetään arvoksi tosi, viestii sovellus käyttäjälle visuaalisen ja animoidun latausikonin avulla sovelluksen latauksen olevan vielä käynnissä. Latausikonin ollessa päällä itse taulukkoelementin tiedot eivät ole tarkasteltavissa eikä muokattavissa. IsBusy-parametri asetetaan todeksi juuri ennen kuin Promise.all-metodi aloittaa rajapintakutsujen käsittelyn. Kun jokainen rajapintakutsu on täyttynyt, palauttaa Promise.all-metodi yksittäisen lupauksen, joka sisältää sarjan käsiteltyjen kutsujen täyttymisarvoja. Riippumatta siitä, täyttyvätkö rajapintakutsut onnistuneina tai epäonnistuneina, asetetaan IsBusy-parametri lopulta takaisin epätodeksi, jolloin käyttöliittymän latausikoni päättyy. Latausikonin päätyttyä käyttäjän on mahdollista joko muokata taulukon tietoja tai nähdä virheilmoitus, mikäli tietojen lataus on epäonnistunut.

Joulea pyydetään muodostamaan tarvittava funktio seuraavalla kehotteella: ”Extend loadHelpers() function in models/Helpers.js to handle loading data from service HELPER_SRV and endpoints ApprovalAreas, MaterialGroups and Plants. Utilize data loading functions implemented in util/oData.js. Call promises must utilize appropriate handleLoadValue... functions implemented in models/Helpers.js. Wrap all calls in Promise.all() which sets IsBusy parameter to false once all calls are fulfilled.”

Joulen luoma vastaus täyttää lähtökohtaisesti kaikki kehotteessa pyydetyt ominaisuudet (Ohjelmakoodi 24). Koodi ei kuitenkaan sellaisenaan ole täysin käyttökelpoinen, sillä esimerkiksi funktiossa määritetty oDataUtil-muuttuja on tuotu Helpers-malliin jo aiemmin. Myös ”var oModel = this” poistetaan, sillä koodin luettavuuden kannalta this-konteksti viittaa selkeämmin tiedoston funktioihin ja ”oModel” taas mallissa hallittavaan dataan.

Ohjelmakoodi 24 Vastauksessa luotu tiedonhakufunktio

```
loadHelpers: function() {
    var oDataUtil = sap.ui.require('bc/thesis/app/util/oData');
    var oModel = this;

    // Set IsBusy parameter to true
    this.setProperty("/IsBusy", true);

    // Define the service name and endpoints
    var serviceName = "HELPER_SRV";
    var endpoints = ["ApprovalAreas", "MaterialGroups", "Plants"];

    // Create an array to store the promises
    var promises = [];

    // Loop through the endpoints and create a promise for each
    endpoints.forEach(function(endpoint) {
        var promise = oDataUtil.read(serviceName, endpoint)
            .then(function(data) {
                // Handle the successful response
                oModel._handleLoadValueHelpDataSuccess("/" + endpoint, data);
            })
            .catch(function(error) {
                // Handle the error response
                oModel._handleLoadValueHelpDataError("/" + endpoint);
            });

        promises.push(promise);
    });

    // Use Promise.all() to wait for all promises to be fulfilled
    Promise.all(promises)
        .then(function() {
            // Set IsBusy parameter to false
            oModel.setProperty("/IsBusy", false);
        });
},
```

Funktio ei sellaisenaan myöskään lataa sovellukseen tietoja, sillä oDataUtil-funktio palauttaa vääriä käsittelijöitä eikä tiedoston kontekstia ole sidottu oikein lupauksen käsittelijöihin. Sovellus ei kaadu, mutta selaimen konsoli-ikkunasta voidaan huomata sovelluksen antama virhe “Cannot read properties of undefined (reading '_handleLoadValueHelpDataSuccess')”. Virheilmoituksen ja kehotteen korjaamisen avulla Joule saadaan sitomaan syntaksi oikein, mutta ei huomioimaan oDataUtil-tiedoston palauttamia käsittelijöitä edes erikseen pyydettäessä. Funktio päädytään korjaamaan loppuun käsin. (Ohjelmakoodi 25)

Ohjelmakoodi 25 Korjattu koodi ja kommentoituna Joulen vastaus

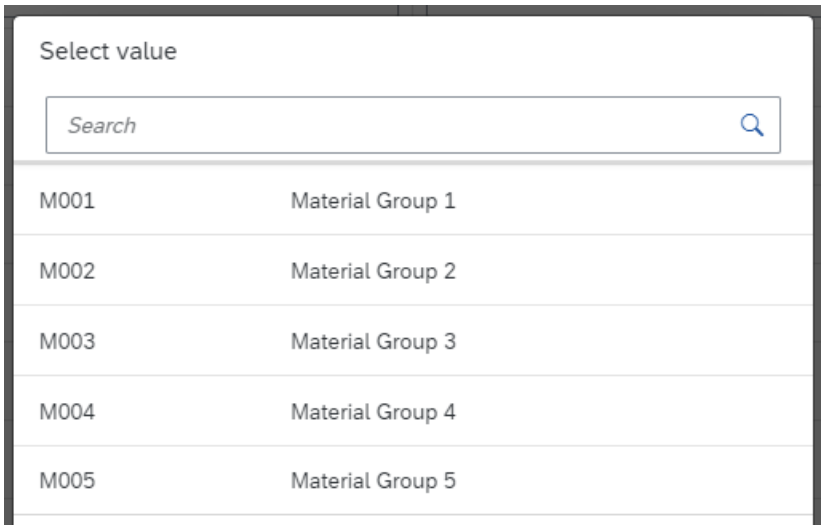
```
// Loop through the endpoints and create a promise for each
endpoints.forEach(function(endpoint) {
    var promise = oDataUtil
        .read(serviceName, '/' + endpoint)
        .done(this._handleLoadValueHelpDataSuccess.bind(this, '/' + endpoint))
        .fail(this._handleLoadValueHelpDataError.bind(this));

    // var promise = oDataUtil.read(serviceName, '/' + endpoint)
    //  .then(function(data) {
    //      // Handle the successful response
    //      this._handleLoadValueHelpDataSuccess('/' + endpoint, data);
    //  }.bind(this, endpoint))
    //  .catch(function(error) {
    //      // Handle the error response
    //      this._handleLoadValueHelpDataError('/' + endpoint);
    //  }.bind(this, endpoint));

    promises.push(promise);
}, this); // Pass 'this' as the second argument to forEach to maintain the correct context
```

Tehtyjen muutosten jälkeen sovelluksesta voidaan nyt aputietoikonia painamalla avata apudialogi, joka esittää listan syötteeseen valittavissa olevia arvoja (Kuva 8).

Kuva 8 Apudialogin lista



6.7 CRUD-toiminnallisuudet

CRUD, eli Create, Read, Update ja Delete, ovat tietojenkäsittelyn neljä perustoimintoa: luo, lue, päivitä ja poista. Sovellukseen on tähän mennessä luotu toiminnallisuus tietojen

lukemiselle. Seuraavissa työvaiheissa lisätään mahdollisuus luoda uusia rivejä, päivittää olemassa olevien rivien tietoja ja poistaa rivejä.

Uusien rivien lisääminen taulukkoon halutaan toteuttaa siten, että käyttäjän painaessa Lisää-painiketta (Add), lisätään taulukon loppuun uusi tyhjä rivi. Käyttäjän tulee pystyä lisäämään taulukkoon useita rivejä, jotta tietojen lisäys olisi sujuvaa. Lisättyjen rivien tietoja ei tallenneta, ennen kuin käyttäjä painaa taulukon alapalkissa sijaitsevaa Tallenna-painiketta (Save).

Taulukosta puuttuu toistaiseksi lisäystoiminnon lisäksi myös Tallenna-painike. Painikkeen XML-koodi ja tapahtumankäsittelyfunktio luodaan projektiin Joulen avulla. Joule myös suoriutuu kehotteesta kerralla oikein, eikä sen luomaa vastausta tarvitse muokata lisää. (Ohjelmakoodi 26).

Ohjelmakoodi 26 Rivin lisäävä funktio

```
onAddPress: function() {
    var oModel = this.getView().getModel("Data");
    var aItems = oModel.getProperty("/Items");
    aItems.push({});
    oModel.setProperty("/Items", aItems);
}
```

Mikäli Joulen vastausta haluttaisiin jatkokehittää, voitaisiin ohjaimen ja mallin erottelua korostaa vielä lisää. Koska Main-ohjaimen on tarkoitus vain hoitaa kommunikaatio näkymän ja mallin välillä, voitaisiin tallennettua dataa muuttava koodi siirtää mallin puolelle ja ohjaimen puolella hoitaa funktion kutsu.

Vaikka sovellukseen on saatu näkyviin aputietodialogit ja käyttäjä voi niistä valita arvon syötteeseen, ei valittu arvo tule näkyviin. Main-ohjaimeen luotujen valueHelpRequest-funktioiden firstValueToSave-parametri viittaa valitun tiedon tallennussijaintiin. Koska sovelluksessa käytettävät tiedot ovat objekteja taulukkomuuttujan sisällä, tarvitaan tallennuspolkuun myös muutettavan objektin indeksisijainti. Joulea pyydetään luomaan painiketapahtumasta indeksisijainnin parsiva funktio. Toimivaa vastausta ei kuitenkaan onnistuta saamaan useidenkaan kehotteiden jälkeen, joten funktio päädytään kirjoittamaan itse.

Viimeisenä CRUD-ominaisuutena lisätään tietojen poistaminen. Aiemmin sovellukseen lisätyt poistoikonit lisätään takaisin käyttöliittymään ja Joulea pyydetään luomaan poistotapahtuman käsittelevä funktio. Toiminnon kohdalla törmätään samoihin ongelmiin kuin tietojen muokkauksen tallentamisen kanssa: Joule ei osaa käsitellä tietorakennetta oikein ja hakea poistettavan rivin indeksisijaintia. Sen luoma koodi ei kuitenkaan ole oikeasta ratkaisusta kaukana, vaan sen suurin ongelma on Joulen käyttämä parametri, jota oEvent-tapahtumaobjektilla ei ole olemassa. Kun Joulelle kertoo, ettei sen kokeilema rowContext-parametriä ole käytettävissä, onnistuu sen korjata koodi toimivaksi. Se ei kuitenkaan viimeistele vastaustaan loppuun asti, vaan jättää koodiin mukaan oRowContext-muuttujan, jota se ei kuitenkaan käytä muualla funktiossa (Ohjelmakoodi 27). Muuttuja poistetaan, mutta vastauksen muihin osioihin ei kosketa, sillä sovelluksen X-ikonia painamalla rivin poisto toimii nyt halutusti.

Ohjelmakoodi 27 Rivin poistava funktio

```
onDeletePress: function(oEvent) {
    var oTable = this.byId('dataTable');
    var oModel = this.getView().getModel('Data');
    var aItems = oModel.getProperty('/Items');
    var oRow = oEvent.getParameter('row');
    var oRowContext = oRow.getBindingContext('Data');
    var iIndex = oTable.indexOfRow(oRow);

    if (iIndex !== -1) {
        aItems.splice(iIndex, 1);
        oModel.setProperty('/Items', aItems);
        sap.m.MessageToast.show('Item deleted successfully');
    } else {
        sap.m.MessageToast.show('Item not found');
    }
},
```

6.8 Muutosten peruminen ennen tallennusta

Koska sovelluksessa tehtyjä tietojen lisäyksiä, muutoksia tai poistoa ei tallenneta ennen Tallenna-painikkeen painamista, voidaan käyttäjälle tarjota mahdollisuus myös tehtyjen muutosten perumiseen Peruuta-painikkeella. Peruutustoimintoa varten sovellukseen on kehitysprosessin alussa luotu valmiiksi painike, jolle implementoidaan oma tapahtumakäsittelijänsä.

Toiminnallisuutta varten sovelluksessa tulee pystyä erittelemään, mitkä ovat tietokannasta alun perin haettuja tietoja ja mitkä muutettuja tietoja. Tätä varten Data-malliin luodaan uusi OriginalItems-taulukkomuuttuja, johon tallennetaan sama tietojoukko, kuin Items-taulukkomuuttujassa. Jotta molempien muuttujien tiedot olisivat itsenäisiä, eivätkä millään tapaa liitoksissa toisiinsa, tulee Items-muuttujan tiedot tallentaa kopiona.

Toiminnallisuudessa tarvittavat asiat kuvaillaan Joulelle. (Ohjelmakoodi 28). Sen vastaus on sinällään toimiva, mutta koodin jäsentelyä korjaillaan käsin siirtämällä OriginalItems-tietojen asetus setItems-funktiosta rajapintakutsujen tuloskäsittelijöihin.

Ohjelmakoodi 28 Tietojen asetus kahteen objektiin

```
setItems: function(aItems) {
    var aItemsCopy = JSON.parse(JSON.stringify(aItems));
    this.setProperty('/Items', aItemsCopy);
},

getOriginalItems: function() {
    return this.getProperty('/OriginalItems');
},

setOriginalItems: function(aItems) {
    this.setProperty('/OriginalItems', aItems);
},

resetItems: function() {
    var aOriginalItems = this.getOriginalItems();
    this.setProperty('/Items', JSON.parse(JSON.stringify(aOriginalItems)));
},

_handleLoadValueHelpDataSuccess: function(aData) {
    this.setItems(aData);
    this.setOriginalItems(aData);
},

_handleLoadValueHelpDataError: function() {
    this.setItems([]);
    this.setOriginalItems([]);
},
```

Jotta käyttäjän ei olisi mahdollista perua tekemiään muutoksiaan vahingossa, pyydetään Joulea luomaan tapahtumankäsittelyfunktio, joka vahvistaa operaation ennen sen suorittamista. Käytännössä Peruuta-painikkeen painaminen näyttää käyttäjälle vahvistusdialogin, jonka avulla tietojen palauttaminen voidaan vielä perua tai vahvistaa. Joulen luoma koodi ei vaadi lainkaan korjauksia, vaan se voidaan ottaa käyttöön sellaisenaan. (Ohjelmakoodi 29).

Ohjelmakoodi 29 Muutosten peruminen ja vahvistusdialogi

```
onCancelPress: function() {
    sap.m.MessageBox.confirm('Are you sure you want to reset the items?', {
        onClose: function(oAction) {
            if (oAction === sap.m.MessageBox.Action.OK) {
                var oDataModel = this.getView().getModel('Data');
                oDataModel.resetItems();
            }
        }.bind(this)
    });
},
```

7 Joulen hyödyllisyyden arviointi

Tässä luvussa käsitellään toteutettua sovelluskehitysprosessia tarkastellen sekä Joulen vahvuuksia että heikkouksia. Luvussa analysoidaan muun muassa Joulen kykyä noudattaa annettuja ohjeita ja kehotteita, sekä sen taitoa tuottaa projektikokonaisuuteen sopivaa ja tarkoituksenmukaista koodia. Lisäksi arvioidaan, kuinka hyvin Joule pystyy seuraamaan JavaScriptin ja SAPUI5:n standardikäytäntöjä sekä nimeämissuosituksia. Tarkastelussa pyritään tarjoamaan kattava kuva siitä, kuinka Joule suoriutuu erilaisista kehitystehtävistä ja missä sen käytössä on mahdollisesti parantamisen varaa.

7.1 Kehotteiden muuntaminen koodiksi

Joulella pystytään luomaan käyttökelpoista koodia, mikäli annetut kehotteet ovat erittäin selkeitä, yksinkertaisia ja kehotteessa pyydettävät muutokset sijoittuvat yhteen tai vain muutamaan tiedostoon. Kokonaisuudessaan kehotteiden tulkitseminen tuntui kuitenkin hyvin ailahtelevalta ja kehotteen uudelleen muotoiluun meni usein enemmän aikaa, kuin koodin kirjoittamiseen ilman Joulea. Vaikka kehotteissa erikseen mainittiin tiedostot, joihin muutoksia haluttiin tehtävän, ei Joule niitä välttämättä huomioinut. Esimerkiksi asynkronisen tiedonhaun työvaiheessa Joule yritti kesken kaiken sijoittaa tiedonhakufunktiot suoraan Main-ohjaimeen, vaikka Helpers-malli oli määritetty kehotteessa ja sinne oli jo luotu alustava tiedonhakufunktio.

Joulen luvataan pystyvän muuntamaan luonnollisella kielellä annetut kehotteet käyttökelpoiseksi koodiksi, mutta suoriutuminen näistä jätti toivomisen varaa etenkin yksityiskohtien huomioimisen suhteen. Keskustelunäkymässä käydyissä keskusteluissa kehotteen muodolla ei huomattu olevan merkitystä Joulen käytökseen. Parhaat tulokset saatiin, kun kehotteet kirjoitettiin pseudokoodimaisina kommentteina suoraan tiedostoon.

Joulen käyttöliittymä sisältää Hyväksy-painikkeen (Accept), jolla vastauksessa luodun koodin voi siirtää automaattisesti suoraan tiedostoon. Painiketta päästiin opinnäytetyössä kuitenkin hyödyntämään harvoin, sillä Joulen vastauskäyttäytyminen teki luodun koodin hyväksymisestä työlästä. Vaikka Joulelta pyydettiin muutoksia vain tiettyyn funktioon, generoi se vastauksessaan vaihtelevasti joko yksittäisen funktion tai koko tiedoston, johon muutokset oli sisällytetty. Vaikka opinnäytetyön aikana käytökseen ei löydetty vaikuttavaa tekijää, tuntui Joule suosivan koko tiedoston palauttamista. Koska kehotteiden tulkinta ei ollut kovin tarkkaa

ja niissä esiintyi jo aiemmin implementoidun koodin muuttamista, olisi koko tiedoston sisältävän vastauksen hyväksyminen vaatinut sen läpi käymistä alusta loppuun asti.

Luotujen koodien korjaaminen annetun virheilmoituksen perusteella onnistui Joulelta kuitenkin poikkeuksetta hyvin. Vaikka se ei siis aina luokaan käyttökelpoista koodia, on Joulesta selkeä apu virheilmoitusten tulkitsemiseen sekä ongelmien korjaamiseen.

7.2 Vastausten tarkoituksenmukaisuus ja tietoisuus

Vaikka Joulen tietoisuuden kattavuudessa on korostettu olevan vielä puutteita, koettiin sen olevan vielä hyvin riittämätön antamaan koko projektin kokonaisuuteen sopivia vastauksia. Vaikka SAP Build Code – Ask Me Anything -lähetyksessä luvattiin Joulen pystyvän esimerkiksi luomaan rakenteita olemassa olevien entiteettien mukaisesti, joutui SAP Fiori -kehityksessä objektien rakenteita toistuvasti tarkentamaan ja erikseen pyytämään niiden noudattamista. Opinnäytetyössä saadun kokemuksen perusteella ei kuitenkaan pystytä sanomaan, olisiko Joulesta saatava hyöty merkittävämpi esimerkiksi backend-kehityksessä tuotettavien tietokantarakenteiden tukena.

Joulen tietoisuudessa huomattiin puutteita kuitenkin myös yksinkertaisemmissa kehitystehtävissä, joissa Joulen olisi voinut olettaa huomioivan yksityiskohtia. Sen lisäksi, että Joule pyrki käyttämään taulukko-ohjainelementille sopimatonta koostetta, jäi siltä myös standardista poikkeavalla etuliitteellä XML-näkymään tuotu kirjastomoduuli huomioimatta. Joule saatiin huomioimaan tiedostoissa määritetyt etuliitteet niistä erikseen mainitsemalla, mutta kehotteeseen annetussa vastauksessaan Joule pyrki virheellisesti lisäämään etuliitteen myös tietojen sitomisen syntaksiin (Ohjelmakoodi 30 Virheellinen etuliite ominaisuussidonnan syntaksissa).

Ohjelmakoodi 30 Virheellinen etuliite ominaisuussidonnan syntaksissa

```
<Select
    items="{c:Tables>/Tables}"
    selectedKey="{c:Tables>/SelectedTable}"
>
    <c:Item key="{c:Tables>Key}" text="{c:Tables>Description}" />
</Select>
```

Selkeä heikennys Joulen käyttöön huomattiin heinäkuun lopulla, kun Joulelle julkaistiin merkittävä UX-päivitys. Opinnäytetyön projektia varten SAP Build Codeen ja Jouleen

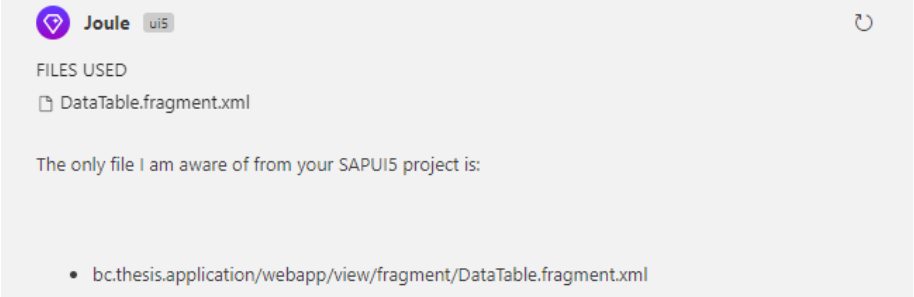
tutustuminen aloitettiin huhti-toukokuussa 2024, jolloin yksi sen ehdottomista vahvuuksista oli kyky luoda annetun tietokantarakenteen mukaista testidataa haluttu määrä ja halutussa tiedostomuodossa.

Heinäkuun päivityksessä kuitenkin muutettiin tapaa, jolle Joulelle annetaan kehotteita. Päivitystä ennen kehotteen pystyi kirjoittamaan vapaassa muodossa. Päivityksen jälkeen jokainen kehote tuli aloittaa kauttaviivakomennolla (slash-command), jota seurasi tarkennus halutun kehotteen tyypistä. Käytettävissä olevat komennot riippuvat siitä, minkälaista projektia kehitetään. (Zapryanov, 2024)

Vaikka SAP Fiori -sovelluskehityksessä käytettävät testipalvelimet ja testidatat ovat SAPUI5:n standardikäytäntöjä, ei testidataa enää päivityksen jälkeen ollut mahdollista luoda halutussa tiedostomuodossa. Oletus on, että varsinainen testidatan luominen oli asetettu /cap-gen-data-komennon alle, mutta mitkään /cap-alkuiset komennot eivät olleet saatavilla SAP Fiori -sovellusta kehittäessä. Koska kehotteen alkuliitteeksi oli pakollista määrittää /ui5, ei Joule suostunut enää luomaan edes erikseen pyydettäessä JSON-muotoista testidataa, vaan se pyrki aina luomaan tiedot taulukkomuodossa sovelluksen malliin tai ohjaimeen. Vaikka tietojen muuntaminen JSON-muotoiseksi ei opinnäytetyöprojektin kokoluokassa ollut erityisen aikaa vievä työvaihe, on kyseessä kuitenkin merkittävä heikennys käyttökokemukseen.

Sovelluskehitysprojektin aikana törmättiin myös ongelmaan, jossa Joule hukkasi tietoisuudestaan kokonaan sen jo aiemminkin muokkaaman Main-ohjaimen. Joulelta myös saatiin erittäin mielenkiintoinen vastaus, kun siltä pyydettiin listausta sen tiedostamista tiedostoista: Joulen mukaan se oli tietoinen ainoastaan taulukko-ohjainelementin sisältävästä XML-näkymästä. (Kuva 9). Vastaus ei kuitenkaan voinut pitää paikkaansa, sillä ennen ja jälkeen kyseistä kysymystä Joule generoi vastauksia, jossa hyödynnettiin myös muita tiedostoja. Koko sovelluskehityksen loppuaikana Main-ohjain ei kuitenkaan palannut Joulen tietoisuuteen, eikä unohdukselle löydetty työn aikana mitään selitystä.

Kuva 9 Joulen lista tiedostetuista tiedostoista



7.3 Käytäntöjen ja nimeämissuositusten noudattaminen

Joulen tuottama koodi noudattaa yleisesti ottaen hyvin SAPUI5:n kehityskäytäntöjä. Esimerkiksi vastauksissa käytetyt muuttujat ovat useimmiten nimetty SAPUI:n suositusten mukaisesti unkarilaista notaatiota (Hungarian notation) hyödyntäen.

Koodissa on välillä havaittavissa elementtejä, jotka eivät täysin noudata ohjelmistokehityksen parhaita käytäntöjä. Näihin kuuluu esimerkiksi kovakoodattujen arvojen käyttö ja toistuvan koodin kirjoittaminen, mitkä voivat heikentää koodin ylläpidettävyyttä ja jatkettavuutta. Opinnäytetyöprojektin laajuuteen Joulen generoimat koodit olivat riittäviä, mutta niiden käyttö sellaisenaan voisi aiheuttaa ongelmia pidemmällä aikavälillä.

Lisäksi Joulen tuottamassa koodissa muuttujien alustaminen harvoin vastaa nykyaikaisia suosituksia. Joule tuntuu selkeästi suosivan vanhempaa var-syntaksia, vaikka modernit JavaScript-suositukset korostavat let- ja const-määreiden käyttöä. Var-määreen käyttö ei ole enää suositeltavaa, sillä se voi aiheuttaa odottamattomia ongelmia muuttujien näkyvyysalueiden (scope) kanssa. Let- ja const-määreet parantavat tietojen hallintaa ja lisäävät koodin turvallisuutta.

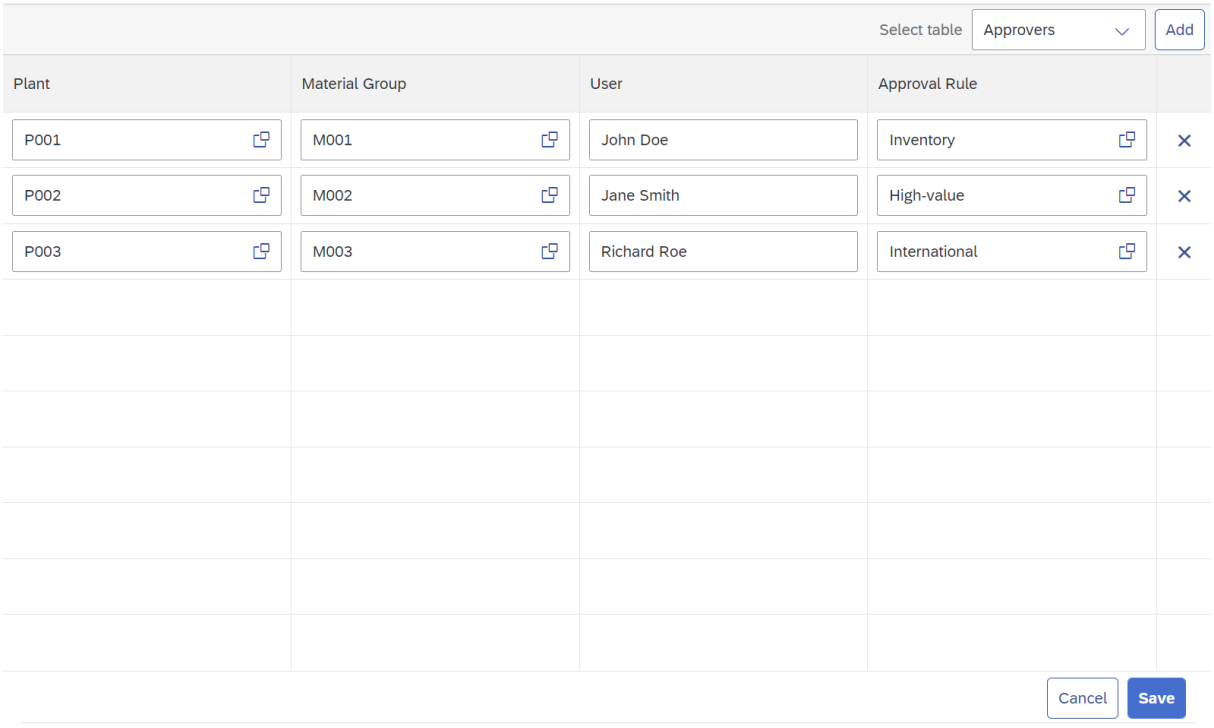
Vaikka Joulen tuottama koodi on monin osin käyttökelpoista, voi vanhentuneiden käytäntöjen suosiminen tuottaa yllättäviä haasteita etenkin koodipohjan laajentuessa. Vanhempaa UI5-versiota käyttävien sovelluksien kohdalla on mahdollista, ettei niissä vielä tueta moderneja let- ja const-määreitä. Opinnäytetyössä kehitettävän sovelluksen kohdalla UI5-versio kuitenkin oli riittävän korkea modernien muuttujien käyttämiseen, jolloin olisi oletettavaa myös Joulen niitä suosivan.

Joulen kanssa työskentely on sujuvaa, kun sen tuottamalle koodille ei aseta tiukkoja tyylisääntöjä. Mikäli yrityksessä käytettävät muotoilusäännöt tai muut koodistandardit asettavat hyvin tiukat rajat tuotettavalle koodille, hidastuu Joulen tuella työskentely huomattavasti. Kehitysympäristöön suositellaankin asetettavan riittävän kattavat tyylisäännöt, jotta pienten muotoseikkojen uudelleenmuotoilu käy mahdollisimman automatisoidusti.

8 Tulokset

Opinnäytetyön tuloksena kehitettiin toiminnallinen SAPUI5-verkkosovellus, joka mahdollistaa taulumuotoisen datan tarkastelun ja muokkauksen (Kuva 10). Käyttöliittymään lisättiin pudotusvalikko, jonka avulla käyttävä voi vaihtaa taulukossa näytettävää tietojoukkoa. Taulukon sarakkeet ja rivit luotiin dynaamisesti SAPUI5:n listasidontaa ja tehdasfunktiota hyödyntämällä. Toteutustapa mahdollistaa useiden tietojoukkojen esittämisen ilman tarvetta luoda jokaiselle taulukolle omaa XML-näkymää. Lisäksi taulukon syötteiden ohjainelementteihin lisättiin aputietoikonit ja -dialogit, joiden käyttämät tietojoukot ladattiin sovellukseen asynkronisesti.

Kuva 10 Valmiin sovelluksen käyttöliittymä



Sovelluskehitysprojektin yhteydessä suoritettiin kattava arviointi Joulesta sovelluskehityksen työkaluna. Joulen toimintaa testattiin monipuolisesti erilaisilla kehotteilla ja komennoilla. Projektin avulla saatiin kokonaisvaltainen kuva siitä, miten Joule tukee kehittäjää erilaisten ohjelmistokehitystehtävien suorittamisessa.

9 Johtopäätökset ja pohdinta

Opinnäytetyön tavoitteena oli selvittää, minkälainen vaikutus SAP:n uusilla työkaluilla, SAP Build Codella ja Joulella, on sovelluskehityksen prosesseihin: kannattaako ne ottaa heti osaksi toimeksiantajayrityksen sovelluskehitystä vai onko työkalujen hyvä antaa vielä kypsyä ja kehittyä.

Opinnäytetyössä toteutetun sovelluskehitysprojektin myötä havaittiin, ettei Joulen taidot vastanneet täysin sille asetettuja odotuksia. Kehotteiden jatkuva hienosäätö ja generoitujen vastauksien virheet hidastivat merkittävästi kehitysprosessia. Vaikka osa virheistä johtuikin kehotteiden epäselvyydestä, olisi SAPUI5-standardimenetelmien, kuten listasidonnan ja ohjainelementtien luomisen, pitänyt toimia ongelmitta. Joulen rajallinen tietoisuus herätti myös huolta.

Joulea hyödyntäessä onkin hyvä kiinnittää erityistä huomiota siihen, ettei virheellistä koodia pääse tuotantoympäristöön. Koska Joulen tekemät virheet ovat ajoittain hyvin huomaamattomia, korostuu sovellusten ja koodin testausprosessien tärkeys entisestään.

Työn aikana todettiin, että vastajulkaistun ja edelleen kehitysvaiheessa olevan alustan arviointi on haastavaa. Tämän vuoksi onkin tärkeää huomioida, ettei työkalujen hyötyä tulevaisuuden sovelluskehityksessä voida täysin ennustaa.

SAP Build Coden ominaisuuksia päästiin opinnäytetyön aikana testaamaan suppeasti. Sen tarjoamat visuaaliset työkalut esimerkiksi projektin alustamiseen ja valmiin sovelluksen julkaisemiseen suoraviivaistavat kehitystyötä kuitenkin huomattavasti. Työkalujen käyttökokemus sovelluskehitystehtävien osalta ei eroa juuri lainkaan lokaalisti asennetun Microsoft Visual Studio Code -koodinmuokkausohjelmassa työskentelystä, mutta niiden selainpohjainen toimintaympäristö voi kuitenkin luoda suurimman haasteen niiden käyttöönotolle. SAP Build Code -sovellusten kehittäminen vaatii aina toimivan verkkoyhteyden, sillä kehitystyö tapahtuu SAP Business Application Studio -alustalla.

Opinnäytetyöprojekti antoi hyvän kuvan siitä, minkälaisia työkaluja SAP Build Code ja Joule ovat. Vaikka niitä ei vielä otettaisikaan käyttöön, on hyvä pitää silmällä sitä, mihin suuntaan tulevaisuudessa julkaistavat päivitykset niitä viedään. Työn jatkosuunnitelmana on selvittää, suoriutuisiko Joule paremmin fullstack-sovelluskehitysprojektin tukena.

10 Yhteenveto

Tutkimuskysymyksiin vastaaminen onnistui kattavasti. Konkreettisen sovelluskehitysprojektin ja sen ohella tehdyillä havainnoilla pystyttiin toteamaan, ettei SAP Build Code ja Joule nykytilassaan vaikuta positiivisesti kehitysprosessin tehokkuuteen tai sujuvuuteen. Joule on parhaimmillaan hyvin yksinkertaisissa tehtävissä ja sen mainostetaankin pystyvän lisäämään näppärästi esimerkiksi painikeohjainelementti XML-näkymään, mutta vastaavat toiminnot sujuvat nykyään nopeasti jo Microsoft Visual Studion automaattisella tekstintäytöllä. Joule noudattaa kohtalaisesti standardien mukaisen koodin luomista, mutta sen vastauksissa esiintyvien virheiden korjaaminen vaatii tietämystä SAPUI5-sovellusten kehittämisestä. Uusien SAP-kehittäjien avuksi Joulea kannattaa käyttää harkiten, sillä sen myötä oppii helposti myös huonoja sovelluskehityskäytäntöjä.

Opinnäytetyöprosessi kasvatti omaa ymmärrystäni sovellusten kehittämisestä SAP Business Application Studio -ympäristössä ja SAP Build Codella. Vaikka Joulen kyvyt jättivätkin toivomisen varaa, on työkaluihin palaaminen tulevaisuudessa helpompaa opinnäytetyön parissa kerätyn kokemuksen myötä. Koen myös vahvuutena sen, että pystyn myöhemmin vertaamaan Joulen kehitystä siihen, mitä se on ollut opinnäytetyön tekohetkellä.

Lähteet

Deloitte. (2022). *State of AI in the Enterprise—5th edition.pdf* [Dataset]. https://www2.deloitte.com/uk/en/pages/deloitte-analytics/articles/state-of-ai-in-the-enterprise-edition-5.html

Glavanovits, R., Koch, M., Krancz, D., & Olzinger, M. (2023). *Full stack development with SAP* (1st edition). Rheinwerk Publishing.

Goebels, C., Modderman, P., Nepraunig, D., & Seidel, T. (2020). *SAPUI5: The comprehensive guide* (2nd edition). Rheinwerk Publishing.

Gupta, R. (2023, helmikuuta 21). *Demystifying SAP Build for Beginners*. SAP Community. https://community.sap.com/t5/technology-blogs-by-sap/demystifying-sap-build-for-beginners/ba-p/13553828

Herzig, P. (2024, kesäkuuta 4). *Joule to Integrate with Microsoft Copilot for a Unified Work Experience*. SAP News Center. https://news.sap.com/2024/06/joule-microsoft-copilot-unified-work-experience/

hurjatron. (2020, huhtikuuta 8). MVC for dummies: Malli, näkymä ja ohjain -arkkitehtuuri web-sovelluksissa. *Hurja Solutions Oy*. https://www.hurja.fi/blogi/mvc-for-dummies-malli-nakyma-ja-ohjain-arkkitehtuuri-web-sovelluksissa/

PRESS, S. (ei pvm.). *What Is SAP Build?* Noudettu 11. elokuuta 2024, osoitteesta https://blog.sap-press.com/what-is-sap-build

SAP. (ei pvm.-a). *Aggregation Handling in XML Views—Documentation*. Noudettu 16. lokakuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/#/topic/19eabf5b13214f27b929b9473df3195b

SAP. (ei pvm.-b). *Binding Types—Documentation*. Noudettu 11. elokuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/sdk/#/topic/91f0d8ab6f4d1014b6dd926db0e91070

SAP. (ei pvm.-c). *Button | SAP Fiori for Web Design Guidelines*. Noudettu 16. lokakuuta 2024, osoitteesta https://experience.sap.com/fiori-design-web/button-web-component/

SAP. (ei pvm.-d). *Data Binding—Documentation*. Noudettu 11. elokuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/sdk/#/topic/68b9644a253741e8a4b9e4279a35c247

SAP. (ei pvm.-e). *Generative AI: A Powerful New Technology*. SAP. Noudettu 31. maaliskuuta 2024, osoitteesta https://www.sap.com/products/artificial-intelligence/what-is-generative-ai.html

SAP. (ei pvm.-f). *List Binding (Aggregation Binding)—Documentation*. Noudettu 16. lokakuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/#/topic/91f057786f4d1014b6dd926db0e91070

SAP. (ei pvm.-g). *Models—Documentation*. Noudettu 11. elokuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/sdk/#/topic/d2c8cf7ae19d447aad5b5ce40e3b14e3

SAP. (ei pvm.-h). *SAP Build Code | Pro-Code App Development | Developer Tools and Services*. SAP. Noudettu 31. maaliskuuta 2024, osoitteesta https://www.sap.com/products/technology-platform/developer-tools.html

SAP. (ei pvm.-i). *SAP History*. SAP. Noudettu 20. maaliskuuta 2024, osoitteesta https://www.sap.com/about/company/history.html

SAP. (ei pvm.-j). *Toolbar Overview | SAP Fiori for Web Design Guidelines*. Noudettu 16. lokakuuta 2024, osoitteesta https://experience.sap.com/fiori-design-web/toolbar-overview/

SAP. (ei pvm.-k). *What is SAP?* SAP. Noudettu 20. maaliskuuta 2024, osoitteesta https://www.sap.com/about/what-is-sap.html

SAP. (ei pvm.-l). *Working with Controls—Documentation*. Noudettu 16. lokakuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/#/topic/91f0a22d6f4d1014b6dd926db0e91070

SAP. (ei pvm.-m). *XML View—Documentation*. Noudettu 11. elokuuta 2024, osoitteesta https://sapui5.hana.ondemand.com/sdk/#/topic/91f292806f4d1014b6dd926db0e91070

SAP. (2019, joulukuuta 3). *The SAP Fiori Design System—Overview and Evolution*. SAP Community. https://community.sap.com/t5/technology-blogs-by-sap/the-sap-fiori-design-system-overview-and-evolution/ba-p/13456112

SAP Community (Ohjaaja). (2024, maaliskuuta 27). *SAP Build Code—Ask Me Anything!* [Video recording]. https://www.youtube.com/watch?v=iMn-VkqsBXw

Solita & IRO Research. (2023). *TOP500 Companies and Generative AI* [Dataset].

Taylor, P. (2024, kesäkuuta 5). *Generative AI Innovations Take Center Stage at SAP Sapphire in 2024*. SAP News Center. https://news.sap.com/2024/06/sap-sapphire-generative-ai-center-stage/

Zapryanov, I. (2024, heinäkuuta 31). *Major UX Update of Joule in SAP Build Code*. SAP Community. https://community.sap.com/t5/technology-blogs-by-sap/major-ux-update-of-joule-in-sap-build-code/ba-p/13770686

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyön sovelluskehitysprojektin aikaiset työvaiheet kirjataan ylös kuvina ja tekstimuotoisina muistiinpanoina. Aineisto käydään läpi ja kirjoitetaan puhtaaksi opinnäytetyötä varten. Koska opinnäytetyössä ei tulla käsittelemään henkilötietoja tai muuta luottamuksellista tai salassa pidettävää aineistoa, säilytetään aineistoja tekijän tietokoneen C-asemalla ja OneDrive-pilvipalvelussa. Valmistunutta aineistoa tullaan säilyttämään 1 vuoden ajan opinnäytetyön hyväksymispäivästä eteenpäin.

Opinnäytetyöaineiston jatkokäyttö työn valmistumisen jälkeen

Tutkimusaineistoa ei jatkokäytetä. Opinnäytetyön tekijä säilyttää aineiston tietoturvallisesti vuoden ajan opinnäytetyön hyväksymispäivästä, jotta opinnäytetyön tulokset voidaan tarvittaessa varmistaa ja hävittää tämän jälkeen aineiston tietoturvallisesti.