

Gabriel Takei

# Lightning-komponenttien uudelleenkäytettävyys

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

1.5.2018

Tekijä Otsikko	Gabriel Takei Lightning-komponenttien uudelleenkäytettävyys
Sivumäärä Aika	35 sivua + 16 liitettä 1.5.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	Yliopettaja Erja Nikunen
<p>Tämän insinööriyön tavoitteena oli tutkia Salesforce-pilvipalvelun Lightning-komponenttien uudelleenkäytettävyttä. Uudelleenkäytettävyden tutkimuksen yhteydessä tavoitteena oli käynnistää Fluido Oy:ssä uudelleenkäytettävyden prosessi. Tarkoituksena oli, että komponentteja pystyisi asentamaan mihin tahansa Salesforce-ympäristöön helposti.</p> <p>Työssä kehitettiin useampi eri Lightning-komponentti, joilla oli erilainen käyttötarkoitus. Komponentit tehtiin täysin dynaamisiksi siten, etteivät ne ole riippuvaisia muista konfiguraatioista tai komponenteista. Kehityksen jälkeen komponentit paketoitiin Salesforcessa ja ne laitettiin Fluidossa yleiseen jakoon komponenttikirjastoon.</p> <p>Työn tavoitteet täyttyivät ja komponentteja käytettiin uudelleen useissa eri projekteissa. Komponentteja asennettiin eri Salesforce-ympäristöihin komponenttikirjaston kautta. Uudelleenkäytettävyden ansiosta komponenttien kehitykseen menevää aikaa pystyttiin vähentämään.</p>	
Avainsanat	Uudelleenkäytettävä, Salesforce, Lightning-komponentti

Author Title	Gabriel Takei Reusability of Lightning Components
Number of Pages Date	35 pages + 16 appendices 1 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Erja Nikunen, Principal Lecturer
<p>The goal of this thesis was to research the reusability of Salesforce Lightning components. While researching the reusability it was also a goal to start a process for reusability in Fluido Oy. The purpose was that components should be easily installed to any Salesforce environment.</p> <p>During the work several Lightning components for different purposes were developed. The components were made completely dynamic, so that they would not be dependent of other configuration or components. After the development, the components were implemented as installable packages in Salesforce to be internally available in Fluidos component library.</p> <p>The goals of this thesis were met as components were reused in many different projects. They were installed into different Salesforce environments via Fluidos component library. Due to the reusability of the Lightning components, time used for development was saved.</p>	
Keywords	Reusability, Salesforce, Lightning component

## Sisällys

### Lyhenteet

1	Johdanto	1
2	Uudelleenkäytettävyys	2
2.1	Hyödyt ja edut	2
2.2	Uudelleenkäytettävyyden haitat	4
2.3	Uudelleenkäytön prosessi	5
3	Fluido Oy	8
4	Teknologiat	9
4.1	HTML5 & CSS3	9
4.2	JavaScript	9
4.3	Salesforce	10
4.3.1	Apex	13
4.3.2	Lightning-komponentit	14
4.3.3	Lightning-komponenttien käyttö	15
4.3.4	Lightning-komponentin kokoelma	16
4.3.5	Peruskomponentit	18
5	Uudelleenkäytettävien Lightning-komponenttien kehitys	20
5.1	Kasaantuva korttielementti	20
5.2	Taulun ja listan sivunumerointi	23
5.3	Dynaaminen datataulu	25
5.4	Datataulukomponentin rakenne ja toiminnallisuus	30
5.5	Testaus	31
6	Uudelleenkäytettävyyden tuomat hyödyt Fluidossa	32
7	Yhteenveto	34
	Lähteet	35
	Liitteet	
	Liite 1. dataTable.cmp -resurssi	
	Liite 2. dataTableController.js -resurssi	
	Liite 3. dataTableHelper.js -resurssi	

- Liite 4. dataTable.design -resurssi
- Liite 5. dataTableTableHeader.cmp -resurssi
- Liite 6. dataTableTableHeaderController.js -resurssi
- Liite 7. dataTableRow.cmp -resurssi
- Liite 8. dataTableRowController.js -resurssi
- Liite 9. dataTableCell.cmp -resurssi
- Liite 10. dataTableCellController.js -resurssi
- Liite 11. dataTablePaginate.cmp -resurssi
- Liite 12. dataTablePaginateController.js -resurssi
- Liite 13. dataTablePageChangeEvent -tapahtuma
- Liite 14. dataTableSortEvent -tapahtuma
- Liite 15. DataTableController.cls
- Liite 16. DataTableControllerTest.cls

## Lyhenteet

CRM	Customer Relationship Management – Asiakkuudenhallinta. Tällä usein viitataan tietojärjestelmiin, joilla hallinnoidaan asiakastietoja ja erilaisia asioita asiakkaisiin liittyen kuten tilauksia.
DML	Data manipulation language. Salesforce.com:in käyttämä datamanipulaatiokieli, jolla voidaan Apex-koodissa lisätä, päivittää tai poistaa tietueita tietokannasta.
MVC	Model-View-Controller. Termi jota käytetään ohjelmoinnissa. Se kuvaa sovelluksen arkkitehtuuria, jossa Model vastaa tietomallia, View-käyttöliittymää ja Controller-kontrolleria, joka toimii mallin ja käyttöliittymän välillä.
SLDS	Salesforce Lightning Design System. Viitekehys, jonka avulla voi tehdä CSS-tyyliin pohjautuvista järjestelmistä Salesforce Lightning -tyylin mukaisia.
SOQL	Salesforce Object Query Language. Salesforce.com:in tietokantakyselykieli, joka palauttaa listan tietueita.
SOSL	Salesforce Object Search Language. Salesforce.com:in tietokantaetsintäkieli, joka palauttaa listan objektien tietuelistoista.
SVG	Skaalautuva vektorigrafiikka (Scalable Vector Graphic) on moderni grafiikkatyyppi, jota käytetään usein nettisivuilla ja sovelluksissa.

## 1 Johdanto

Sovelluksien kehityksessä puhutaan usein uudelleenkäytettävyydestä. Uudelleenkäytettävyydellä pyritään yleensä säästämään aikaa, rahaa tai resursseja. Terminä uudelleenkäytettävyys on hyvin laaja käsite, ja se voi tarkoittaa eri ihmisille eri asioita. Tuote- ja sovelluskehityksessä uudelleenkäytettävyys olisi aina hyvä määritellä. Tärkeätä on tietää, mikä on uudelleenkäytettävää ja miten sitä käytetään uudelleen. Uudelleenkäytettävyyden tueksi ja edistämiseksi sille olisi näin ollen hyvä perustaa oma prosessi.

Salesforce on yksi maailman johtavimmista pilvipalvelutuottajista. Jokaisella Salesforcen asiakkaalla on oma ympäristö. Asiakkaat voivat tehdä erilaisia asetuksia ja muutoksia omaan ympäristöön. Ympäristöön on myös mahdollista kehittää omia sivuja ja applikaatioita. Salesforce on kehittänyt viitekehityksen nimeltä Lightning. Se pyrkii vastaamaan moderneja viitekehityksiä kuten Angularia tai Reactia. Lightning-viitekehityksessä asiakkaat voivat luoda omia sovelluksen osia, joita kutsutaan komponenteiksi. Komponenteista voidaan kasata isompia sovelluksia. Lightning-komponenttien yksi päätarkoitus on, että ne ovat uudelleenkäytettäviä.

Tämän insinööriyön tavoitteena oli kehittää Lightning-komponenttien uudelleenkäytettävyyttä. Kehitykseen valittiin useampi erilainen komponentti, joiden käyttötarkoitus ja kohderyhmä ovat toisistaan poikkeavia.

Tämä insinööriyö on toiminut perustana Fluidon uudelleenkäytettävyyden prosessille. Työn tuloksena ja tekijän aktiivisuudesta Fluidoon perustettiin uudelleenkäytettävyyden kirjasto. Kirjastoon voi jokainen kehittäjä tallentaa lähdekoodit ja asennuslinkit teke miinsä komponentteihin. Näin valmiita tai muokattavissa olevia komponentteja voidaan ottaa mihin tahansa käyttöön nopeasti.

## 2 Uudelleenkäytettävyys

Lombard Hill Group määrittelee, että sovelluksen uudelleenkäytettävyys on sitä, kun käytetään olemassa olevia resursseja. Resursseja voivat olla muun muassa tuotteet ja sivutuotteet, jotka sisältävät sovellusosia, testejä, suunnitelmia ja dokumentaatioita. Uudelleenkäytettävyys on tehokkainta silloin, kun se on systemaattista. Uudelleenkäyttö on systemaattista silloin, kun uudelleenkäytettävyyden prosessi on hyvin suunniteltu ja elinkaari määritelty ja sitä tuetaan, resursoidaan ja sen käyttöä kannustetaan. (Lombard Hill Group 2017.)

Douglas C. Schmidt määrittelee systemaattisen uudelleenkäytettävyyden lisäksi opportunistisen uudelleenkäytettävyyden. Opportunistisella uudelleenkäytettävyydellä tarkoitetaan muun muassa sitä, kun kehittäjä kopioi olemassa olevasta koodista osia uuteen koodiin. Opportunistinen uudelleenkäytettävyys toimii rajoitetusti yksittäiselle kehittäjälle tai pienelle ryhmälle, mutta se ei ole tarpeeksi toimiva tapa isoille organisaatioille tai useille ryhmille. (Douglas C. Schmidt 2017.)

Uudelleenkäytettävyyttä voidaan tehdä eri tavoilla. Koostuvalla tavalla uusia sovelluksia luodaan käyttämällä uudelleen olemassa olevia resursseja. Kun taas tuottavalla tavalla uusia sovelluksia luodaan korkean tason määritelmistä hyödyntämällä sovelluskehittämiä. (Lombard Hill Group 2017.)

Hyödyntäminen on sitä, kun käytetään olemassa olevia resursseja ja niitä muokataan uusien tarpeiden mukaan. Mikäli hyödyntäminen on hallittu hyvin, se voi olla edullisempaa kuin sovelluksen tekeminen tyhjältä pöydältä säästämällä aikaa ja vaivaa. Hyödyntäminen voi myös johtaa huonompaan sovelluksen laatuun riippuen tarvittavista muutoksista, joita voi olla vaikeata istuttaa jo olemassa olevaan uudelleenkäytettävään sovellukseen. Pahimmassa tapauksessa sovelluksesta tulee monta eri versiota, jotka lisäävät ylläpitoa. (Lombard Hill Group 2017.)

### 2.1 Hyödyt ja edut

Yrityksillä on nykyisin enemmän paineita saada tuotteet nopeammin markkinoille samalla vähentämällä sovellusten kehitykseen meneviä kustannuksia ja aikaa ja parantaen



niiden laatua. Monien yritysten tuotteet perustuvatkin uudelleenkäytettävyyteen saavuttaakseen strategiset tavoitteet. (Lombard Hill Group 2017.)

Uudelleenkäytettävyys voi parantaa yrityksen tuottavuutta. Ensimmäisen investoinnin jälkeen sovellusten kehitykseen ja ylläpitoon käytetty aika vähenee. Uudelleenkäytettävän resurssin käytettävyyden arviointi vie kuitenkin aikaa, mutta se on yleensä paljon vähemmän kuin kehitykseen menevä aika. (Journal of Theoretical and Applied Information Technology 2018.) Hewlett-Packard on raportoinut, että heidän tuottavuutensa on kasvanut kuuden ja 40 prosentin välillä, kun projekteja on uudelleenkäytetty. Myös tuotteiden saaminen markkinoille nopeutuu. (Lombard Hill Group 2017.)

Uudelleenkäytettävyyden avulla projekteja voidaan aikatauluttaa tarkemmin. Mikäli uudelleenkäytettäviä resursseja on hyödynnetty aikaisemmissa projekteissa, projektipäälliköt voivat katsoa edellisten projektien dokumenteista uudelleenkäytettävän resurssin käyttöönottoon menneen ajan ja työmäärän. Uudelleenkäytettävien resurssien jatkuvan uudelleenkäytön avulla käyttöönottoon menevä aika standardoituu ja samalla työmääräarviot tarkentuvat. (Journal of Theoretical and Applied Information Technology 2018.)

Sovellusten laatu paranee uudelleenkäytön myötä. Uudelleenkäytetyt sovellukset sisältävät yleensä vähemmän virheitä kuin täysin uudelleen kirjoitetut koodit. Hewlett-Packardilla huomattiin, että uudelleen käytetyissä koodeissa virheitä ilmeni kymmenen kertaan vähemmän kuin uudessa koodissa. Lisäksi HP:n laatu on parantunut 24 ja 76 prosentin välillä. (Lombard Hill Group 2017.)

Jokainen peräkkäinen uudelleenkäyttö testaa uudelleenkäytettävän resurssin uudelleen ja näin ollen virheitä esiintyy harvemmin. Peräkkäiset uudelleenkäytöt lisäävät uudelleenkäytettävän resurssin luotettavuutta ja hyödyllisyyttä ja samalla pienentäen riskejä. (Journal of Theoretical and Applied Information Technology 2018.)

Sovellusten, joissa uudelleen käytetään samoja komponentteja, voidaan olettaa toimivan ja käyttäytyvän yhdenmukaisesti. Esimerkiksi käyttöliittymän toiminallisuudet kaikissa sovelluksissa käyttäytyvät samantapaisesti. Näin esimerkiksi käyttäjät muistavat paremmin, miten sovelluksen osat toimivat. (Lombard Hill Group 2017.)

Uusien sovellusten prototyyppien testaus on nopeampaa, sekä käyttäjävaatimukset voivat tulla helpommin esille, kun sovelluksen komponentit ovat valmiiksi saatavilla. Tämä

testaus tuo myös sovelluksessa olevat ongelmat sen kehityksen elinkaaren aikana nopeammin esille. Näin ongelmiin voidaan vaikuttaa varhaisemmassa vaiheessa välttämättä suuret korjauskulut. (Lombard Hill Group 2017.)

Jos jokin sovellus sisältää jo tarvittavat toiminallisuudet, niin sitä uudelleenkäyttämällä riskit pienenevät. Tämä yleensä vaatii sen, että sovelluksella on jo valmis rajapinta integraatioita varten. Uudelleenkäytön ansiosta aikaa jää myös enemmän uudelleenkäytettävän osan parantamiseen. (Lombard Hill Group 2017.)

## 2.2 Uudelleenkäytettävyyden haitat

Uudelleenkäyttö ei ole kaikkialla tuonut mitään suuria parannuksia laadussa tai tuottavuudessa. Monissa hienostuneissa viitekehyksissä on osoitettu, että uudelleenkäytettävyys on hyödyllistä. Näissä tapauksissa kyseessä on kuitenkin yleensä pieni aihealue, kuten jokin koodikirjasto tai graafiset käyttöliittymät. Teoriassa yritykset tunnustavat uudelleenkäytettävyyden tuomat hyödyt ja sen käytön palkitsemisen. Käytännössä sen toteuttaminen on kuitenkin erittäin haastavaa monista eri tekijöistä johtuen. Usein yrityksillä, joilla on paljon vanhoja järjestelmiä ja niiden kehittäjiä, on haasteita uudelleenkäytettävyyden hyödyntämisessä. Suurimmat syyt eivät yleensä ole teknisiä, kuten organisaatiosta, taloudellisista tai hallinnollisista johtuvat esteet. (Journal of Theoretical and Applied Information Technology 2018.)

Kehitys, käyttöönotto ja uudelleenkäytettävyyden tukeminen vaativat syvää ymmärrystä kehittäjien tarpeista ja liiketoiminnan vaatimuksista. Organisaatiosta johtuvat esteet tulevat esille, kun kehittäjien ja uudelleenkäytettävien resurssien hyödyntävien projektien määrä kasvaa. Näissä tapauksissa on vaikeaa muodostaa organisaatio, joka tukisi tehokkaasti uudelleenkäytettävyyttä antamalla palautetta sen hyödyntämisestä. (Journal of Theoretical and Applied Information Technology 2018.)

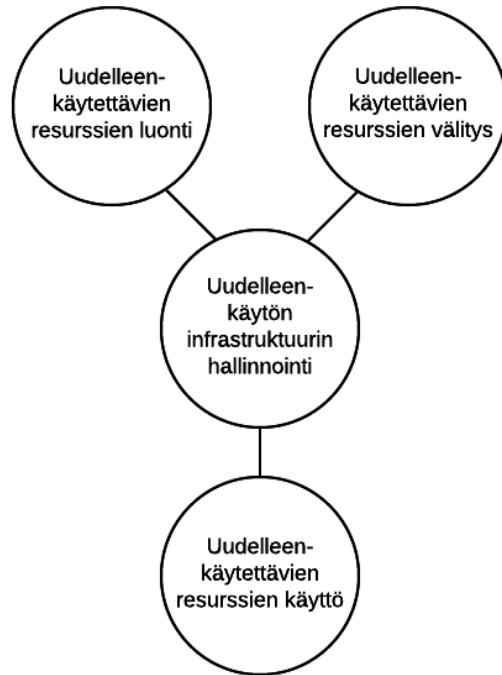
Uudelleenkäytettävyyden hyödyntäminen tehokkaasti vaatii usein taloudellista investointia. Alussa investointi on yleensä suuri, koska uudelleenkäytettävyys täytyy määritellä ja mahdollisesti pitää palkata lisää työntekijöitä. Yksi rajoittava tekijä saattaa olla myös se, jos uudelleenkäytettävyyttä tukeva organisaatio on omana kustannuspaikkanaan. Tällöin veloituksien ja takaisinmaksujen perustaminen saattaa olla vaikeaa. (Journal of Theoretical and Applied Information Technology 2018.)

Hallinnollisista esteistä johtuen suurien organisaatioiden useiden liiketoimintayksiköiden välillä on vaikeaa luetteloida, arkistoida ja hakea uudelleenkäytettäviä resursseja. Vaikka on yleistä, että kehittäjät kaivaa muista sovelluksista uudelleenkäytettäviä koodia, niin kehittäjien on kuitenkin vaikeaa löytää sopivia uudelleenkäytettäviä resursseja heidän työskentely-ympäristön ulkopuolelta. (Journal of Theoretical and Applied Information Technology 2018.)

Usein on myös mahdollista, että uudelleenkäytettävyydestä ei ole hyötyä, mikäli kehittäjällä ei ole tarpeeksi teknisiä taitoa hyödyntää tai luoda uudelleenkäytettäviä resursseja. Usein kehittäjillä ei ole kykyä tai kokemusta oman aihealueen perussuunnittelumalleista, jonka takia heidän on vaikea luoda tai käyttää uudelleenkäytettäviä resursseja tehokkaasti. (Journal of Theoretical and Applied Information Technology 2018.)

### 2.3 Uudelleenkäytön prosessi

Uudelleenkäyttö ei ole ainoastaan sitä, että luodaan varasto, johon sovelluksia tallennetaan ja josta niitä hyödynnetään. Uudelleenkäyttö on toteutettu parhaiten silloin, kun sitä tukee prosessi. Uudelleenkäytön prosessi voi yksinkertaisimmillaan olla sellainen, joka sisältää vain neljä aktiviteettia. Nämä aktiviteetit ovat uudelleenkäytön infrastruktuurin hallinnointi sekä uudelleenkäytettävien resurssien luominen, välitys ja käyttö (kuva 1). (Lombard Hill Group 2017.)



Kuva 1. Uudelleenkäytön prosessin aktiviteetit (Lombard Hill Group 2017)

Infrastruktuurin hallinnoinnin tarkoituksena on asettaa uudelleenkäytön säännöt ja tavoitteet, jotka tukevat uudelleenkäyttöä. Infrastruktuurin hallinnointi on koko prosessin keskeisin aktiviteetti. Siinä asetetaan standardit, joiden mukaan hyväksytään uudelleenkäytettävien resurssien lisäys, muokkaus ja poisto kirjastosta. Hallinnoinnissa resursoidaan ja aikataulutetaan uudelleenkäyttöön menevien komponenttien käyttöönotto ja samalla komponenttien tavoitteet asetetaan vastaamaan liiketoiminnan tavoitteita. Hallinnointi myös asettaa uudelleenkäytön kannustimet ja uudelleenkäytön taloudelliset mallit. (Lombard Hill Group 2017.)

Uudelleenkäytettävien resurssien luomisessa kehitetään, luodaan ja uudelleen suunnitellaan resursseja tavoitteiden mukaisesti. Tähän aktiviteettiin kuuluu myös prosessi, jolla tunnistetaan ja analysoidaan eri järjestelmien samanlaisuuksia ja eroavaisuuksia. Näiden tulosten perusteella aktiviteetissa voidaan luoda erilaisia komponentteja uudelleenkäyttöä varten. (Lombard Hill Group 2017.)

Välitysaktiviteetilla autetaan uudelleenkäyttöä sertifoimalla, säätämällä, ylläpitämällä, tukemalla, mainostamalla ja välittämällä uudelleenkäytettäviä resursseja. Tämä aktiviteetti sisältää myös luokittelun ja haun resursseihin uudelleenkäyttökirjastossa. (Lombard Hill Group 2017.)

Uudelleenkäytettävien resurssien käyttö tapahtuu silloin, kun järjestelmä muodostuu uudelleen käytettävistä resursseista. Käyttäjät hyödyntävät heille saatavilla olevia resursseja kirjastosta ja integroivat niitä järjestelmiinsä samalla antaen palautetta olemassa olevista ja uusista tarvittavista resursseista. (Lombard Hill Group 2017.)

### 3 Fluido Oy

Fluido Oy on vuonna 2009 perustettu yritys, jonka koko liiketoiminta perustuu Salesforce.com-järjestelmän ympärille. Keskeisin asia Fluidolle on kuitenkin aina asiakas, kuten Salesforce. Fluido toteuttaa monia erilaisia Salesforce-projekteja asiakkailleen, kuten käyttöönottoja, kustomointeja, migraatioita, integraatioita, koulutuksia jne. Fluido on kasvanut lyhyessä ajassa Pohjoismaiden johtavimmaksi Salesforce-partneriksi, ja se työllistää reilu 200 henkeä yhteensä viidessä maassa. Fluidolla on myös Pohjoismaiden ainoat sertifioidut kouluttajat.

Fluido tekee pääsääntöisesti asiakasprojekteja. Tämä tarkoittaa sitä, että lähes kaikki, mitä asiakkaille tehdään, on heille räätälöityjä ratkaisuja. Tämän työn aikana Fluidon liiketoimintaan ei kuulunut tuotekehitystä. Räätälöityjen ratkaisujen uudelleen hyödyntäminen muihin projekteihin on usein hankalaa, koska ne on tehty aina asiakkaan vaatimusten mukaisesti. Tästä johtuen Fluidolla ei myöskään ole ollut prosesseja uudelleenkäytettävyydelle.

Ilman uudelleenkäytön prosessia monia samantapaisia komponentteja tehdään usein uudelleen. Haasteena on se, kun kehittäjät eivät välttämättä tiedä, että sama komponentti on jo olemassa toisen tekemänä. Toisena haasteena on se, että monet komponentit tehdään asiakkaiden ympäristöön, johon kaikilla kehittäjillä ei ole pääsyä. Asiakkaiden ympäristöjä on satoja, ja niiden kaikkien läpikäynti komponenttien etsimiseksi vie aikaa. Usein uuden projektin yhteydessä konsultit eivät välttämättä ole myöskään tietoisia siitä, että toiseen projektiin tehty komponentti voisi olla uudelleenkäytettävissä ilman tarvetta sille, että kehittäjät ohjelmoisivat komponentin uudelleen.

Lightning-komponenttien tultua on kuitenkin mahdollista tehdä niinkin pieniä komponentteja, joita voidaan uudelleen käyttää lähes jokaisessa projektissa. Lightning-komponenteista kerrotaan tarkemmin luvussa 4.

## 4 Teknologiat

Tässä työssä tehdyt sovelluskomponentit ovat nettiselainkäyttöön pohjautuva. Sovellukset on tehty Salesforce.com-ympäristössä, ja ne koostuvat muun muassa HTML5-, CSS3- ja JavaScript-ohjelmointikielistä.

### 4.1 HTML5 & CSS3

Verkkosivut koostuvat yleensä sivuista, joiden taustalla on merkintäkieli. HTML eli hypertekstin merkintäkieli (Hypertext Markup Language) on eräänlainen ohjelmointikieli, jonka avulla voidaan kehittää erilaisia verkkosivuja ja sovelluksia. Sivut koostuvat useista eri elementeistä, jotka toimivat sivujen rakennusosina. Elementit ovat merkintäkielessä kirjoitettu HTML-tunnisteilla. Tunnisteet pitävät sisällään sivulla olevaa sisältöä, esimerkiksi otsikko tai tekstikappale. Käyttäjä ei suoraan näe HTML-tunnisteita sivun piirtyessä selaimeen, vaan selain käyttää tunnisteita piirtääkseen sivun. (w3schools 2017.)

HTML on kehittynyt vuosien aikana ja uusin versio on HTML5. Sen edut aikaisempiin versioihin on se, että sen kanssa voidaan käyttää moderneja SVG-grafiikkaelementtejä sekä erilaista multimediaa kuten videoita ja ääniä. (w3schools 2017.)

CSS (Cascading Style Sheets) on ohjelmointikieli, joka määrittelee HTML-dokumentin tyylin. Se määrittelee miltä HTML-sivu tulisi näyttää. CSS voi määrittellä muun muassa erilaisten elementtien värejä, kokoja, näkyvyyttä ja animaatioita. CSS:n uusin versio on nimeltään CSS3. (w3schools 2018.)

### 4.2 JavaScript

JavaScript esitettiin ensimmäistä kertaa vuonna 1995 tapana lisätä sovelluksia nettisivuille Netscape Navigator -nettiselaimella. JavaScript on vuosien aikana kehittynyt, ja nykyään kaikki graafiset nettiselaimet tukevat sitä. Se mahdollistaa modernien nettisivujen ja websovellusten luomisen, joiden käyttö on sulavaa eikä sivun tarvitse lataantua uudelleen jokaisesta toiminnasta. Sen avulla voidaan luoda monenlaisia interaktiivisia sivuja, ja se mahdollistaa erinomaisen käyttökokemuksen luomisen. (Haverbeke 2017.)

JavaScriptistä on useita versioita, joista uusin on ECMAScript 2018. Nimi tuli Ecma International -organisaatiosta, joka teki standardoinnin. Pisimpään käytössä ollut versio oli ECMAScript 3, jota käytettiin laajalti vuosien 2000 – 2010 aikana. Uudemmat versiot ovat tulleet tämän jälkeen lähes vuosien välein, ja ne ovat sisältäneet paljon uusia ominaisuuksia. (Haverbeke 2017.)

### 4.3 Salesforce

Salesforce on amerikkalainen ohjelmistoyritys, jonka Marc Benioff perusti kehittäjien Parker Harrisin, Dave Moellenhoffin ja Frank Dominguezin kanssa vuonna 1999. Yrityksen tarkoituksena oli uudistaa tapaa, jolla yritykset hoitavat päivittäisiä tapahtumia asiakkaidensa kanssa. Salesforce kehitti pilvipalveluna asiakkuudenhallintajärjestelmän, joka on vuosien aikana kehittynyt valtavaksi sovellusalustaksi. Järjestelmän kehityksen yhteydessä Salesforce pyrkii aina hyödyntämään uusimpia teknologioita. Fokuksena on kuitenkin aina asiakas, ja siksi järjestelmää pyritään kehittämään siten, että yritykset pysyvät paremmin lähestymään asiakkaitaan. (Salesforce.com 2017.)

Sovellusalustana Salesforce muodostuu useasta eri sovelluksesta, joita ovat Sales Cloud, Service Cloud, Marketing Cloud, Analytics Cloud, Community Cloud, App Cloud ja IoT Cloud (kuva 2). Jokaisella sovelluksella on eri käyttötarkoitus sekä lisenssit, mutta niitä kuitenkin käytetään ja hallinnoidaan saman alustan kautta ja yhdessä ne muodostavat koko Salesforcen pilvipalvelun. (Salesforce.com 2017.)



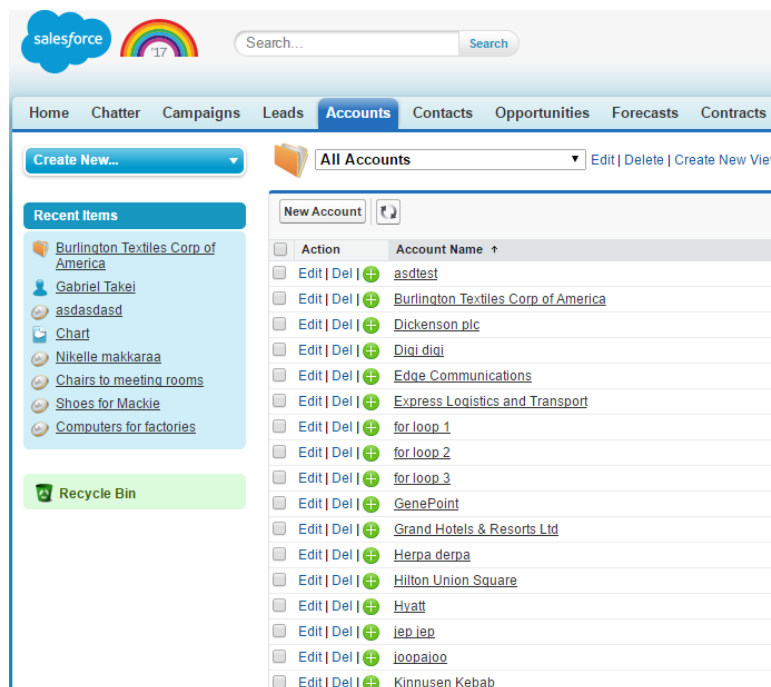
Kuva 2. Salesforce-pilvialustan sovellukset

Salesforcessa on vakiona tietomalli, jossa puhutaan objekteista (object) ja tietueista (record). Objektit ovat tietokantatauluja, joilla on useita kenttiä (field), ja tietueet ovat yhden objektin ilmentymiä. Vakiona Salesforcessa on yli 600 objektia ja jokaisella objektilla on



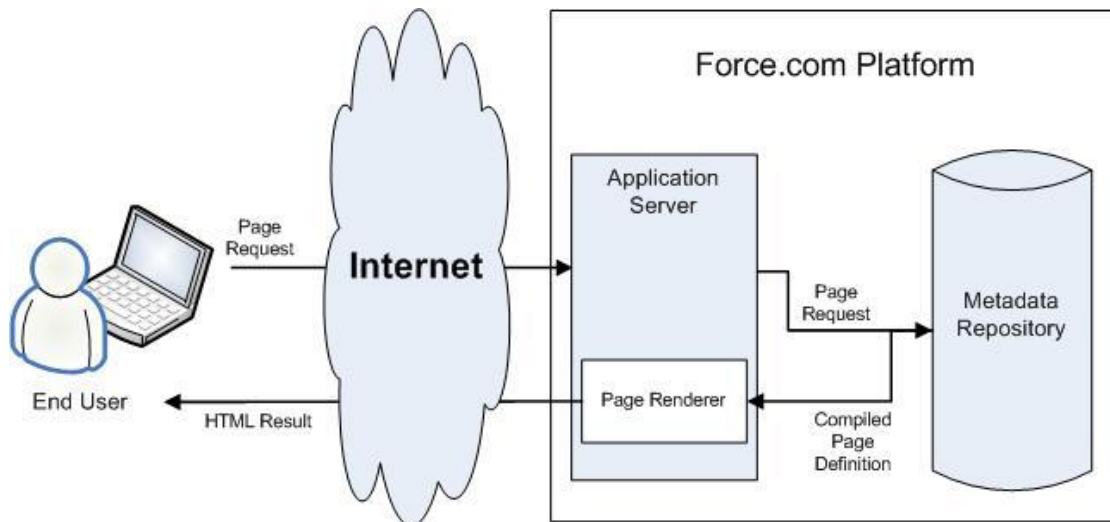
myös omat vakiokentät. Järjestelmävalvojat pystyvät luomaan Salesforceen myös omia objekteja ja kenttiä. Omia kenttiä voi lisätä myös vakio-objekteihin. Tietomallissa esimerkiksi asiakastiedoille hyödynnetään vakio-objektia nimeltä Account. Jokaisen asiakkaan tiedot tallennetaan kyseiseen objektiin tietueina. Vakiokenttinä asiakasobjektilla ovat mm. nimi, osoite ja yhteyshenkilö. (Salesforce.com 2017.)

Järjestelmää käytetään nettiselaimella joko tietokoneella, tabletilla tai puhelimella. Sisäänkirjautuneet käyttäjät näkevät Salesforcea objektit välilehtinä (tab), joita klikkaamalla käyttäjä pääsee kyseisen välilehden sivulle (kuva 3). Välilehden sivun ei tarvitse välttämättä olla objektin sivu. Se voi olla myös täysin objektista riippumaton oma sivu. (Salesforce.com 2017.)



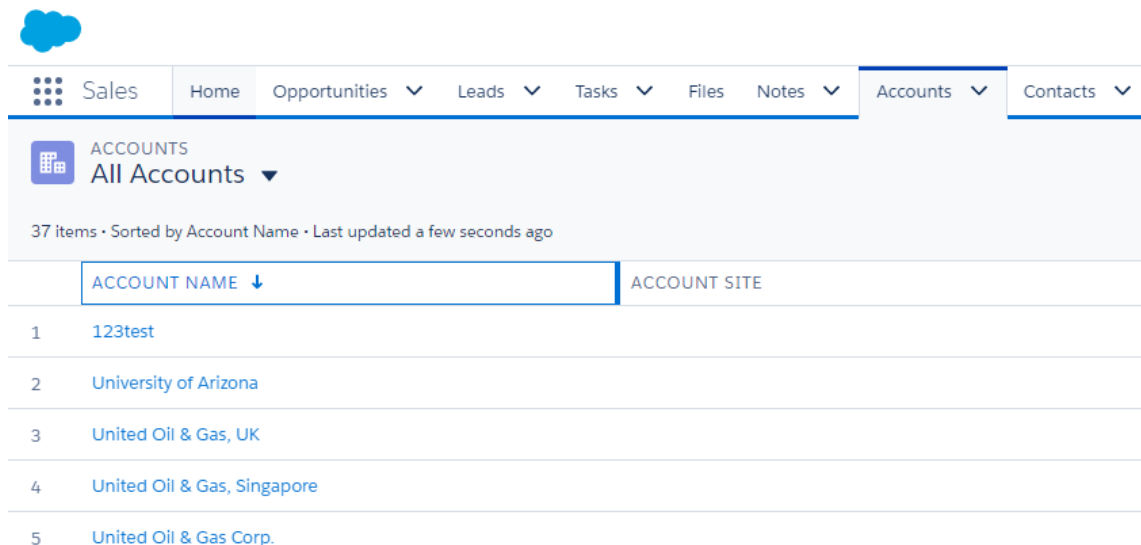
Kuva 3. Salesforceen perusnäkökulma

Salesforcen käyttöliittymä ja sivut perustuvat Visualforce-sivuihin, jotka ovat verrattavissa HTML-sivuihin. Visualforce-sivut ovat MVC-mallin mukaisia ja niillä voi olla vakio-kontrolleri tai täysin oma kontrolleri. Kontrollerit ovat oliomallin mukaisia Apex-ohjelmointikielellä toteutettuja luokkia. Visualforcen kontrollerit ovat tilallisia ja niiden käytönaikainen tila on palvelimen puolella (kuva 4). (Salesforce.com 2017.)



Kuva 4. Visualforce-arkkitehtuuri (Salesforce.com 2017)

Salesforce julkisti uuden käyttöliittymäteknologian vuonna 2014 (kuva 5). Uusi teknologia on nimeltään Lightning. Lightning-viitekehys on komponenttipohjainen ratkaisu, jossa Visualforce-sivujen sijasta on Lightning-sivuja. Lightning-sivuja pystyy muokkaamaan työkalulla nimeltä Lightning App Builder. Lightning-sivut koostuvat Lightning-komponenteista, joita kehittäjät voivat ohjelmoida ja järjestelmänvalvojat voivat uudelleen käyttää App Builderissa. Uuden käyttöliittymän tyyli on nimeltään Lightning Design System (SLDS), jota voi myös käyttää Visualforce-sivuilla. Nämä kaikki muodostavat yhdessä käyttökokemuksen, jota Salesforce kutsuu Lightning Experienceksi. (Salesforce.com 2017.)



	ACCOUNT NAME ↓	ACCOUNT SITE
1	123test	
2	University of Arizona	
3	United Oil & Gas, UK	
4	United Oil & Gas, Singapore	
5	United Oil & Gas Corp.	

Kuva 5. Lightning-käyttöliittymä

Visualforce sekä Lightning pystyvät hyödyntämään Apex-kielellä ohjelmoituja kontroleita kommunikoitaessa palvelimen kanssa. Visualforcella erillisen kontrollerin luominen ei ole tarpeen, mikäli Salesforcen omia vakiokontrollereita voidaan hyödyntää. Tämä on yleensä Visualforcessa tapauskohtainen riippuen sivun vaatimuksista. Lightning-komponentit vaativat erillisen Apex-kontrollerin, mikäli niiden tarvitsee saada tai muokata useamman kuin yhden tietueen dataa Salesforcesta. (Salesforce.com 2017.)

#### 4.3.1 Apex

Apex on olio-ohjelmointikieli, jonka syntaksi näyttää Javalta. Se toimii kuin tietokannan toiminnot. Apex mahdollistaa ohjelmoijien lisätä business-logiikkaa moniin eri järjestelmän tapahtumiin, esimerkiksi napin painalluksiin. Apexia voidaan myös kutsua Web-palveluilla ja objektikohtaisista laukaisimista. Laukaisimet ovat Apex-koodia, jotka käynnistyvät automaattisesti riippuen, miten ne on määritelty. Laukaisimet voivat laueta ennen tai jälkeen, kun tietuetta luodaan, päivitetään tai poistetaan. (Salesforce.com 2017.)

Apex sisältää sisäänrakennettuja toimintoja ja sillä voidaan muun muassa tehdä seuraavat toimenpiteet:

- Datamanipulaatio kutsut lisäävät, päivittävät tai poistavat tietueita tietokannasta.

- Tietokantakyselyt ja -haut SOQL (Salesforce Object Query Language) ja SOSL (Salesforce Object Search Language) auttavat palauttamaan listan tietueista.
- Silmukoiden (loop) ja toistorakenteiden avulla voidaan prosessoida useita tietueita kerralla.
- Lukitussyntaksin avulla voidaan estää tietueiden päivityskonfliktit silloin kun kaksi tai useampi käyttäjä yrittää muokata tietuetta samaan aikaan.
- Voidaan tehdä API-kutsuja metodeista, jotka on tallennettu toisiin Apex-luokkiin.
- Voidaan tehdä varoituksia ja virheilmoituksia, kun käyttäjä yrittää muuttaa tai poistaa objektia tai kenttää, joihin viitataan Apex-koodissa.

Näiden lisäksi Apexia on helppo käyttää, koska se perustuu samankaltaiseen syntaksiin kuin Java. Sen muuttujat ja ilmaisut, lohkot ja ehdolliset lausunnot, toistorakenteet, oliot ja taulut ovat lähes identtiset kuin Javassa. (Salesforce.com 2017.)

Apex on dataan kohdistunut kieli. Se on suunniteltu suorittamaan useita tietokantakutsuja ja DML-toimintoja yhdellä kerralla. Se on voimakkaasti kirjoitettu kieli, joka käyttää suoria viittauksia objekteihin ja kenttiin. Apex epäonnistuu koodin koonnissa, mikäli yksikin viittaus on virheellinen. Apex on täysin tulkittu, suoritettu ja kontrolloitu Salesforce.com-pilvialustassa. Apexin suoritin on suunniteltu siten, että se suojelee alustaa muun muassa muistivuodoilta kuten ikuisilta silmukoilta. Jokaisella palvelimella saattaa olla useamman eri asiakkaan ympäristöjä. Joten on erittäin tärkeätä, ettei toisen asiakkaan virheellinen Apex-koodi aiheuta muistivuotoja, jotka saattaisivat vaikuttaa palvelimen suorituskykyyn hidastaen muiden ympäristöjä. (Salesforce.com 2017.)

Jokaista Apex-luokkaa varten tulisi olla sitä vastaava testiluokka, joka pitää sisällään yksikkötestit. Salesforcen tuotantoympäristöön ei voi siirtää Apex-koodia kehitysympäristöistä ellei niitä varten ole tehty yksikkötestejä. Yksikkötestien tulee kattaa vähintään 75 % Apex-koodista, mikäli se halutaan viedä tuotantoympäristöön. (Salesforce.com 2017.)

#### 4.3.2 Lightning-komponentit

Komponentit ovat itsenäisiä kokonaisuuksia, jotka edustavat uudelleenkäytettäviä käyttöliittymän osia. Ne voivat olla pieniä sisältäen esimerkiksi ainoastaan yhden painikkeen, tai ne voivat olla isoja kokonaisia sovelluksia. Lightning-viitekehys sisältää valmiiksi luotuja vakiokomponentteja, joita järjestelmänvalvojat voivat suoraan käyttää luodessaan

omia Lightning-sivuja. Kehittäjät voivat ohjelmoida myös omia komponentteja, joita voidaan käyttää Lightning-sivuilla ja myös muissa komponenteissa. (Salesforce.com 2017.)

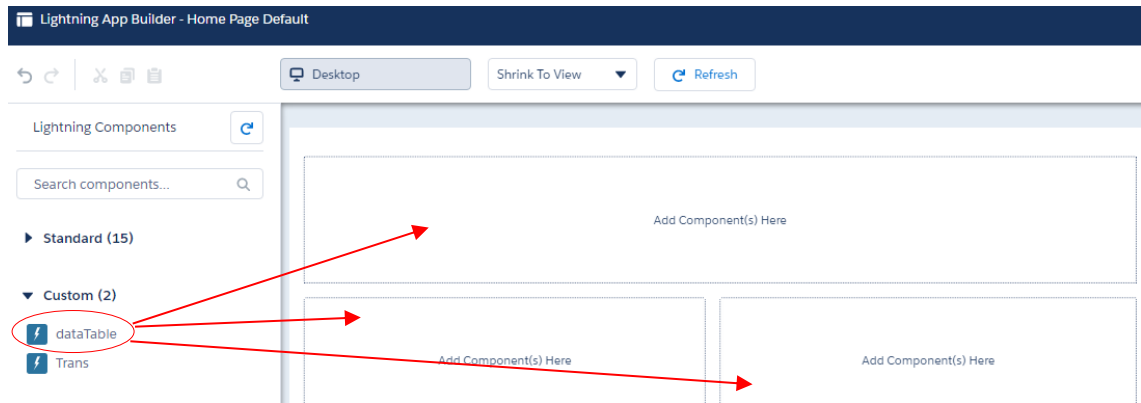
Lightning-komponentit pohjautuvat avoimen lähdekoodin Aura-viitekehikseen, joka on tarkoitettu dynaamisten sovellusten luontiin laitteesta riippumatta. Lightning-komponentit piirtyvät nettiselaimen HTML DOM -elementteinä, ja ne tukevat muun muassa HTML5-, CSS3- sekä JavaScript-kieliä. Lightning-komponentit ovat moderneja käyttäjäpohjaisia ratkaisuja, ja niiden istunnon aikainen yhteys palvelimeen on tilaton ja käyttäjän puoli tilallinen. Näin logiikkaa ja laskentaa on siirretty palvelimelta käyttäjän puolelle. (Salesforce.com 2017.)

Lightning-komponentit ovat tapahtumaohjattuja, ja ne pystyvät kommunikoimaan toistensa kanssa tapahtumien avulla. Komponentteihin pystyy määrittelemään, mikä käyttöliittymässä tapahtuva vuorovaikutus laukaisee tapahtuman. Vastaavasti komponenttiin pystyy määrittelemään, mitä tapahtumaa se käsittelee ja mitä tapahtuu, kun tapahtuma vastaanotetaan. Lightning-viitekehiksessä voi tehdä kahdenlaisia tapahtumia: komponentti- ja applikaatiotapahtumia. Komponenttitapahtumia pystyvät käsittelemään ainoastaan komponentit, jotka laukaisivat tapahtuman tai komponentit, joiden sisäiset komponentit laukaisivat tapahtuman. Applikaatiotapahtumia pystyvät käsittelemään kaikki samalla sivulla olevat komponentit. Tapahtumat pystyvät myös kuljettamaan erilaisia arvoja sen laukaisijalta sen kuuntelijalle. Esimerkiksi jos käyttäjä valitsee valintalistalta arvon, ja siitä laukaistaan tapahtuma. Käyttäjän valitsema arvo voidaan kuljettaa tapahtumalla sen kuuntelijalle. (Salesforce.com 2017.)

#### 4.3.3 Lightning-komponenttien käyttö

Lightning-komponentteja voidaan käyttää Salesforcessa muun muassa Lightning-sivuilla, Community-sivuilla, Visualforce-sivuilla sekä ulkoisilla verkkosivuilla. Näiden lisäksi komponentteja voidaan käyttää toisten komponenttien koodissa. Tällöin puhutaan sisäkkäisistä komponenteista. (Salesforce.com 2017.)

Lightning-sivuja tehdessä ja muokatessa järjestelmänvalvojat voivat käyttää Lightning App Builderiä. App Builder on graafinen käyttöliittymä, jossa komponentteja voidaan raahata haluamiin paikkoihin (kuva 6). Tämän lisäksi joillakin komponenteilla voi olla erilaisia arvoja, joita järjestelmänvalvojat voivat syöttää ja niillä vaikuttaa esimerkiksi komponentin käyttäytymiseen tai tyyliin. (Salesforce.com 2017.)



Kuva 6. Näkymä Lightning App Builderistä, jossa komponentteja voidaan raahata sivulle.

Salesforce Community on yhteisösivusto, jonka järjestelmänvalvojat voivat luoda organisaationsa ympäristöön. Community-sivustot on tarkoitettu organisaation ulkopuolisille käyttäjille, kuten asiakkaille tai partnereille. Kuten Lightning-sivuilla, myös Communityllä on oma graafinen työkalu sivujen luomista varten. Työkalun nimi on Community Builder, joka on lähes identtinen Lightning App Builderin kanssa komponenttien osalta. (Salesforce.com 2017.)

Mikäli Lightning-komponentteja halutaan käyttää Visualforce-sivuilla tai ulkoisilla verkkosivuilla, niin sitä varten Salesforce on kehittänyt tekniikan nimeltä LightningOut. Käytännössä tarvittavaa komponenttia varten täytyy luoda erillinen Lightning-aplikaatio, jonka kautta avataan rajapinta komponentille. Visualforce-sivulle tai ulkoiselle verkkosivulle upotetaan JavaScript-koodi, jossa kutsutaan tätä kyseistä applikaatiota LightningOut-kirjaston avulla. Näin Lightning-komponentin saa vaikka yrityksen omille kotisivuille. (Salesforce.com 2017.)

Lightning-komponentteja voi myös myydä ja jakaa muille Salesforcen asiakkaille Salesforcesta löytyvästä palvelusta nimeltä AppExchange. Viedäkseen komponentin AppExchangeen, tulee komponentin läpäistä Salesforcen turvatarkastukset. Yleensä tämä ei ole nopea tai helppo prosessi. (Salesforce.com 2017.)

#### 4.3.4 Lightning-komponentin kokoelma

Yksittäinen Lightning-komponentti on kokoelma resursseja, jotka yhdessä muodostavat komponentin (taulukko 1).

Taulukko 1. Lightning-komponenttikokoelman resurssit ja tiedostonimet

Resurssi	Resurssinimi	Käyttö
Komponentti tai applikaatio	esimerkki.cmp tai esimerkki.app	Kokoelman ainoa vaadittu resurssi, joka sisältää komponentin merkintäkielen. Kokoelmassa voi olla vain yksi komponentti tai applikaatio resurssi.
CSS-tyyli	esimerkki.css	Sisältää komponentin tyylin.
Kontrolleri	esimerkkiController.js	Sisältää käyttäjä puolen kontrollerin metodit ja tapahtumien käsittelyn.
Design	esimerkki.design	Tarvittava tiedosto, jos komponenttia käytetään App Builderilla, Lightning-sivuilla tai Community Builderilla.
Dokumentaatio	esimerkki.auradoc	Sisältää kuvauksen, esimerkkikoodia ja viittauksia esimerkkikomponentteihin.
Piirtäjä (Renderer)	esimerkkiRenderer.js	Käytetään kun halutaan käyttää muuta kuin komponentin vakiopiirtäjää.
Avustaja (Helper)	esimerkkiHelper.js	Sisältää JavaScript-funktioita joita kaikki komponentissa oleva JavaScript -koodit voivat kutsua.
SVG-tiedosto	esimerkki.svg	Oma svg-ikoni komponenteille, joita käytetään App Builderissa tai Community Builderilla.

Komponentin ainoa vaadittu resurssi on komponentti- tai applikaatioresurssi. Kyseinen resurssi on tiedosto, joka sisältää komponentin merkintäkielen. Merkintäkielessä mainitaan kaikki komponentin muuttujat ja tapahtumat, joita se käsittelee tai laukaisee. Nämä on yleensä mainittu merkintäkielessä ennen varsinaisia käyttöliittymän osia. Käyttöliittymän elementit voi merkintäkielessä kirjoittaa usealla eri tavalla. Kielinä voi käyttää peruskomponentteja, HTML5-kieltä tai muita Lightning-komponentteja. (Salesforce.com 2017.)

Lightning-komponentin käyttäjän puolen logiikka kirjoitetaan komponentin kontrolleriin ja avustajaan. Kontrolleri ja avustaja ovat JavaScript-tiedostoja, ja ne sisältävät funktioita. Kontrolleriin kirjoitetaan funktiot, joita käyttöliittymä kutsuu suoraan, ja avustajaan kirjoitetaan muita funktioita, joita kontrolleri kutsuu. Kontrollerissa käsitellään tapahtumat,

joita komponentin on määritelty kuuntelemaan. Kontrollerissa määritellään myös, mitä komponentti tekee, ennen kuin se piirtyy selaimeen. (Salesforce.com 2017.)

Jokaiselle Lightning-komponentille voi myös määrittellä Apex-kontrollerin. Apex-kontrolleri on tarpeen, mikäli Lightning-komponentin tarvitsee saada tai viedä dataa tietokantaan. Apex-metodeja pystyy kutsumaan Lightning-komponentin kontrollerista tai avustajasta. Lightning-komponentin ollessaan tilaton palvelimeen nähden kaikki komponentin kutumat Apex-metodit tulee olla staattisia. Apex ei ole kuitenkaan osa komponentin kokoelmaa, eli jos komponentin esimerkiksi poistaa, niin Apex jää silti vielä järjestelmään. Komponenteilla voi olla vain yksi Apex-kontrolleri määriteltynä, mutta jokaisella sisäkkäisellä komponentilla voi olla myös oma Apex-kontrolleri. Lightning-komponentilla ei välttämättä tarvitse olla mitään käyttöliittymässä näkyvää elementtiä. Se voi esimerkiksi sisältää vain metodikutsuja. Tällaisia komponentteja kutsutaan palvelukomponenteiksi. Ne käytännössä mahdollistavat ylemmän tason komponentille useamman Apex-kontrollerin. (Salesforce.com 2017.)

Design-tiedosto on erittäin olennainen osa komponentin kokoelmaa, kun puhutaan komponentin uudelleenkäytettävyydestä. Design-tiedostossa määritellään, mitä muuttujien arvoja Salesforce-järjestelmänvalvojat voivat komponentille antaa, kun he käyttävät komponenttia Lightning App Builderissä tai Community Builderissä. Näiden avulla voidaan mahdollistaa dynaamisia komponentteja, jotka toimivat eritavoin riippuen, mitä arvoja komponentille annetaan. (Salesforce.com 2017.)

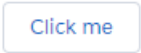
#### 4.3.5 Peruskomponentit

Peruskomponentit ovat Salesforcen kehittämiä komponentteja, joita muut kehittäjät voivat käyttää omien komponenttien merkintäkielessä. Peruskomponentteja ei voi muokata, sillä ne ovat Salesforcen hallinnoimia eikä niitä näe esimerkiksi järjestelmän konfiguraatiossa. (Salesforce.com 2018.)

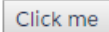
Peruskomponentteja on monenlaisia ja niiden olennaisin asia on se, että ne helpottavat ja nopeuttavat omien komponenttien kehitystä. Salesforce on kehittänyt jo yli 90 peruskomponenttia. Peruskomponentit voivat olla esimerkiksi DOM-elementtejä, kuten nappi. Kehittäjä voi siis itse valita, haluaako hän käyttää peruskomponenttia vai HTML-nappia (kuva 7). (Salesforce.com 2018.)



```
<lightning:button label="Click me" onclick="{!c.handleClick}" />
```



```
<button onclick="{!c.handleClick}">  
  Click me  
</button>
```



Kuva 7. Esimerkki peruskomponenttinapin ja HTML-napin erosta.

Peruskomponentit, jotka piirtyvät DOM-elementteinä sisältävät automaattisesti Salesforce Lightning -tyylin. Tyyli päivittyy dynaamisesti, mikäli Salesforce tekee tyyleihin muutoksia. Peruskomponentit myös lataantuvat paljon nopeammin, sillä ne ovat jo valmiiksi ladattu käyttäjälle, eivätkä ne vaadi ylimääräistä prosessointia. (Salesforce.com 2018.)

Osa peruskomponenteista mahdollistaa datan näyttämisen ja muokkaamisen ilman Apex-kontrolleria. Apex-luokkien kirjoittamisen pois jättäminen nopeuttaa huomattavasti komponenttien kehittämistä. Se myös nopeuttaa niiden viemistä tuotantoon, sillä pakollisia yksikkötestejä ei tarvitse tehdä, mikäli käytössä ei ole Apex-kontrolleria. (Salesforce.com 2018.)

## 5 Uudelleenkäytettävien Lightning-komponenttien kehitys

Yleisesti uudelleenkäyttö moderneissa sovelluksissa on erittäin haastavaa. Kokonaisten sovellusten arkkitehtuuri saattaa muuttua asiakkaiden vaatimusten johdosta. Joissakin tapauksissa uuden sovelluksen luominen olisi vähemmän työlästä kuin uudelleenkäytettävän sovelluksen muuttaminen. Isojen yhden sivun sovelluksien sijaan komponenttisuuntautuneisuus tekee uudelleenkäytettävyydestä helpompaa, koska sovelluksia voidaan pilkkoa pienempiin komponentteihin ja niitä voi uudelleen käyttää eri tavoin.

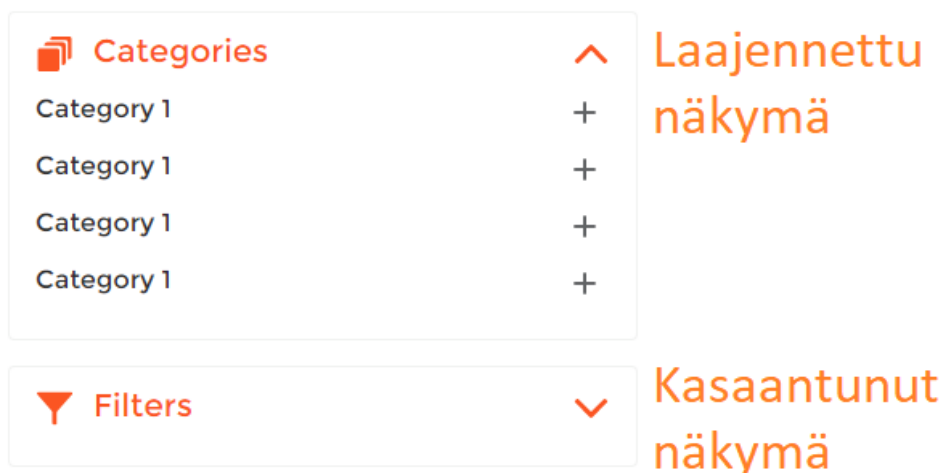
Tämän työn aikana on toteutettu useampi uudelleenkäytettävä Lightning-komponentti:

- kasaantuva korttielementti
- taulun ja listan sivunumerointi
- dynaaminen datataulu.

Kaikki tehdyt komponentit ovat Fluidon kirjastossa jaossa sisäisesti. Kirjastosta komponentin voi asentaa asiakkaan ympäristöön. Tämän jälkeen niitä voi käyttää sellaisinaan tai muokata niitä tarvittaessa asiakkaan tarpeen mukaan.

### 5.1 Kasaantuva korttielementti

Salesforcen Lightning-käyttöliittymässä on visuaalinen määrite nimeltä kortti (Card). Kortti on käytännössä tila, jota ympäröi kehys. Salesforce on kehittänyt kortille myös oman peruskomponentin. Kortti vie kuitenkin näytöltä sen sisällön verran tilaa, ja joissakin tapauksissa käyttäjät voivat haluta piilottaa kortin sisällön. Tätä varten työssä on tehty pieni komponentti, joka käytännössä laajentaa Salesforcen peruskomponenttia lisäämällä toiminallisuuden piilottaa kortin sisältö (kuva 8).



Kuva 8. Esimerkki kasaantuvasta korttikomponentista.

Kasaantuva korttielementti on hyvin yksinkertainen komponentti, ja se koostuu vain kolmesta tiedosta. Komponentin resurssissa merkintäkieli pitää sisällään Salesforceen kortti peruskomponentin, jolle asetetaan otsikko ja ikoni sekä sen sisällön paikka (koodi 1). Merkintäkieli pitää myös sisällään logiikan, joka näyttää erilaisen kuvakkeen riippuen, onko kortti laajennettu vai kasattu.

```
<aura:component>
  <aura:attribute type="String" name="header" />
  <aura:attribute type="Boolean" name="isOpen" />
  <aura:attribute type="String" name="iconName" />
  <aura:handler name="init" action="{!c.doInit}" value="{!this}" />
  <lightning:card >
    <aura:set attribute="title">
      <lightning:layout horizontalAlign="spread">
        <div>
          <aura:if isTrue="{!empty(v.iconName)}">
            <lightning:icon iconName="{!v.iconName}" size="small"
              class="slds-p-right_small catIcon"/>
          </aura:if>
          {#v.header}
        </div>
        <div class="collapseButton" onclick="{!c.handleClick}">
          <aura:if isTrue="{!v.isOpen}">
            <lightning:icon iconName="utility:chevronup" size="small"
              class="catIcon"/>
          <aura:set attribute="else">
            <lightning:icon iconName="utility:chevrondown" size="small"
              class="catIcon"/>
          </aura:set>
        </aura:if>
      </div>
    </lightning:layout>
  </aura:set>
  <div aura:id="catBox" class="slds-p-horizontal_medium">
    {!v.body}
  </div>
</lightning:card>
</aura:component>
```

Koodi 1. Kasaantuvan korttikomponentin merkintäkielen resurssi.

Komponentin kontrollerissa on vain kaksi funktiota. Toisella funktiolla alustetaan halutut tyyli luokat ja toista puolestaan kutsutaan silloin, kun kortin pienennys tai laajennus kuvaketta painetaan ja se vaihtaa samalla kortin tilaa (koodi 2).

```

({
  doInit : function(cmp, evt, helper){
    var el = cmp.find("catBox");
    if(cmp.get("v.isOpen")){
      $A.util.removeClass(el, "slds-is-collapsed");
      $A.util.addClass(el, "slds-is-expanded");
    } else{
      $A.util.addClass(el, "slds-is-collapsed");
      $A.util.removeClass(el, "slds-is-expanded");
    }
  },
  handleClick : function(cmp, evt, helper){
    var el = cmp.find("catBox");
    cmp.set("v.isOpen", !cmp.get("v.isOpen"));
    $A.util.toggleClass(el, "slds-is-collapsed");
    $A.util.toggleClass(el, "slds-is-expanded");
  },
})

```

Koodi 2. Kasaantuvan korttikomponentin JavaScript-kontrolleri.

Lisäksi komponentti sisältää tyyli tiedoston, jossa asetetaan värit kortin otsikolle ja kuvakkeelle.

Kasaantuvan korttikomponentin uudelleenkäytettävyys on rajattu siten, että sitä voi käyttää muiden komponenttien merkintäkielessä. Sitä ei voi sellaisenaan käyttää esimerkiksi App Builderissa, koska komponentille pitää jotenkin määritellä sisältö. Kortille voi määrittellä attribuutteina otsikon, ikonin sekä sen, onko kortti alustuksessa laajennettu vai kasaattuna. Kortin sisältö laitetaan merkintäkielessä kortin aloitus ja lopetus merkkien väliin (koodi 3).

```

<c:SCatFiltBox header="Categories" isOpen="true" iconName="utility:overflow">
  <c:SCategoryItem isOpen="true"/>
  <c:SCategoryItem isOpen="false"/>
  <c:SCategoryItem isOpen="false"/>
  <c:SCategoryItem isOpen="false"/>
</c:SCatFiltBox>

```

Koodi 3. Kasaantuvan korttikomponentin käyttö merkintäkielessä.

## 5.2 Taulun ja listan sivumerointi

Tauluihin ja listoihin yleensä liittyy jokin tapa, millä niiden datan lataamiseen ei mene paljoa aikaa. Tämä toiminallisuus on usein tehty sivumeroinnilla. Sivumeroinnissa on yleensä painikkeet seuraavalle ja edelliselle sivulle. Joissakin sivumeroinneissa on myös numeropainikkeet, joilla voi suoraan valita jonkin sivun.

Tämän työn yhteydessä on tehty sivumerointikomponentti, joka tarjoaa kehittäjille visuaalisen elementin sivumerointiin. Komponentti itsessään ei vaihda dataa tai sivua, mutta se vaihtaa näytettävän sivun numeroa, jota ylemmän tason komponentti puolestaan voi käyttää oikean datan näyttämiseen.

Työssä tehty sivumerointikomponentti näyttää painikkeet edelliselle ja seuraavalle sivulle. Lisäksi komponentti näyttää aina maksimissaan 5 numeroa keskitetysti. Valittu sivu on aina keskellä, ellei se ole ensimmäinen, toinen, toiseksi viimeinen tai viimeinen sivu (kuva 9).



Kuva 9. Sivumerointikomponentti

Sivumerointikomponenttia käytetään muiden komponenttien merkintäkielessä. Se ei ole tietoinen datasta, jota on tarkoitus sivuttaa. Komponentille annetaan parametreina arvot sivujen määrästä ja sivusta, jota halutaan näyttää (koodi 4). Sivua vaihtaessa komponentti päivittää näytettävän sivun numeroa.

```
<c:Pagination totalPages="20" currentPage="{!v.pageToShow}" />
```

Koodi 4. Sivumeroinnin käyttö merkintäkielessä.

Sivumerointi koostuu neljästä tiedostosta. Sen merkintäkieli on yksinkertainen ja pitää sisällään napit edelliselle ja seuraavalle sivulle sekä numerolistan näytettävistä sivuista (koodi 5).

```

<aura:component>
  <aura:attribute type="Integer" name="currentPage" default="1" />
  <aura:attribute type="Integer" name="totalPages" default="20" />
  <aura:attribute type="Integer[]" name="pageButtons" />
  <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
  <lightning:buttonGroup >
    <aura:if isTrue="{!v.currentPage != 1}">
      <lightning:button onclick="{!c.previous}">
        <lightning:icon iconName="utility:chevronleft" size="x-small"/>
      </lightning:button>
    </aura:if>
    <aura:iteration items="{!v.pageButtons}" var="page">
      <lightning:button label="{!page}" onclick="{!c.changePage}"
        class="{!if(v.currentPage == page,
          'pageButton currentPage', 'pageButton')}"/>
    </aura:iteration>
    <aura:if isTrue="{! v.currentPage != v.totalPages}">
      <lightning:button onclick="{!c.next}">
        <lightning:icon iconName="utility:chevronright" size="x-small"/>
      </lightning:button>
    </aura:if>
  </lightning:buttonGroup>
</aura:component>

```

#### Koodi 5. Sivunumeroinnin merkintäkieli

Komponentin kontrolleri sisältää vain neljä funktiota. Alustava funktio asettaa komponentin numeropainikkeet, ja loput funktiot ovat sivun vaihtamista varten edelliselle, seuraavalle tai valitulle sivulle (koodi 6).

```

({
  doInit : function(cmp, evt, helper){
    helper.calculatePageButtons (cmp);
  },
  previous : function(cmp, evt, helper){
    cmp.set("v.currentPage", cmp.get("v.currentPage")-1);
    helper.calculatePageButtons (cmp);
  },
  changePage : function(cmp, evt, helper){
    var newPage = evt.getSource().get("v.label");
    if(newPage !== cmp.get("v.currentPage")){
      cmp.set("v.currentPage", newPage);
      helper.calculatePageButtons (cmp);
    }
  },
  next : function(cmp, evt, helper){
    cmp.set("v.currentPage", cmp.get("v.currentPage")+1);
    helper.calculatePageButtons (cmp);
  },
})

```

#### Koodi 6. Sivunumeroinnin kontrolleri.

Avustajatiedosto pitää sisällään yhden funktion, joka laskee sivun vaihtojen yhteydessä uudet numerot numeropainikkeille. Funktiota kutsutaan kontrollerista (koodi 7). Komponentin neljäs tiedosto on tyylitiedosto, joka pitää sisällään ainoastaan värimäärityksen painikkeille.

```

({
  calculatePageButtons : function(cmp){
    var pages = cmp.get("v.totalPages");
    var currentPage = cmp.get("v.currentPage");
    var pageButtons = [];
    if(currentPage <= 3){
      for(var i = 0; i < 5; i++){
        if((i+1)<=pages){
          pageButtons.push(i+1);
        }
      }
    } else{
      if(currentPage === 4 && currentPage === pages){
        pageButtons.push(currentPage-3);
      } else if(currentPage === pages){
        pageButtons.push(currentPage-4);
        pageButtons.push(currentPage-3);
      } else if((currentPage+1)===pages){
        pageButtons.push(currentPage-3);
      }
      pageButtons.push(currentPage-2);
      pageButtons.push(currentPage-1);
      pageButtons.push(currentPage);
      if((currentPage+1)<=pages){
        pageButtons.push(currentPage+1);
      }
      if((currentPage+2)<=pages){
        pageButtons.push(currentPage+2);
      }
    }
    cmp.set("v.pageButtons", pageButtons);
  }
})

```

Koodi 7. Sivunumeroinnin avustaja.

### 5.3 Dynaaminen datataulu

Ideota siitä, millaisia uudelleen käytettäviä komponentteja Lightning-tekniikalla voisi tehdä, on lähes rajaton määrä. Tähän työhön työläimmäksi komponentiksi pyrittiin valitsemaan sellainen, jota käytetään lähes jokaisessa projektissa, joissakin jopa useaan kertaan. Datataulu osoittautui tekijän omasta kokemuksesta sellaiseksi komponentiksi, jota käytetään lähes jokaisessa projektissa. Lisäksi tämän työn aikana Salesforce ei ollut vielä julkaissut omaa dynaamista komponenttia datataulua varten.

Järjestelmänvalvojat voivat App tai Community Builderissä käyttää komponenttia raahaten sen haluamalleen sivun osalle. Kun järjestelmävalvoja on raahannut komponentin haluamaansa paikkaan, hän voi sen jälkeen asettaa datataululle erilaisia arvoja (kuva 10). Kyseiset arvot on määritelty koodissa komponentin Design-tiedostossa (liite 4) ja ne ovat riippuvaisia komponentin vastaavista attribuuteista komponenttiedostossa (liite 1). Insinööritöön teon aikana komponentille voi antaa seuraavia arvoja:

- Title – Otsikko, joka tulee taulun yläpuolelle.
- sObject – objekti (tietokannan taulu), josta data halutaan.
- fields – Kentät joita tauluun halutaan sarakkeiksi.
- clause – ehtolauseke joka on käytännössä SOQL -lauseke.
- initial sort by – sarake, jonka mukaan taulukko on alustavasti järjestetty.
- records per page – määrä kuinka monta riviä näytetään per sivu.
- show records per page picklist – valinta siitä, näytetäänkö käyttäjälle valintalista, jolla voidaan vaihtaa rivien määrää per sivu.
- records per page picklist values – lista arvoista, jotka tulevat rivien määrä valintalistaan.
- display icon/link to records – valinta siitä, näytetäänkö ensimmäisellä sarakkeella ikoni, joka toimii myös linkkinä kyseisen tietueen sivulle.
- related object – mikäli komponenttia halutaan käyttää toiseen objektiin liittyvän datan näyttöön, voidaan tähän laittaa sen objektin nimi. Esimerkiksi jos datataulu komponenttia käytetään Account tyyppisen tietueen sivulla ja taululla näytetään Opportunity -tietueita, mutta halutaan näyttää vain kyseiseen Accountiin liittyvät tietueet.



Page > dataTable

---

Title i

sObject (FROM) \*REQUIRED\* i

fields (SELECT) \*REQUIRED\* i

clause (WHERE) i

Initial sort by \*REQUIRED\* i

Records per page \*REQUIRED\* i

Show records per page picklist i

Records per page picklist values i

Display icon/link to record i

Related object field name i

Kuva 10. Attribuuttien syöttö Lightning-komponenttiin Lightning App ja Community Builderissä

Tulevaisuudessa arvoja voi olla enemmän, jotka lisäävät komponentin muokattavuutta ja monimutkaisuutta. Arvojen syötön jälkeen sivu tallennetaan ja käyttäjät voivat käyttää kyseistä taulua (kuva 11).

Hello

NAME ↑	CREATED DATE
University of AZ SLA	03. kesäk. 2016
University of AZ Portable Generators	03. kesäk. 2016
United Oil Standby Generators	03. kesäk. 2016
United Oil SLA	03. kesäk. 2016
United Oil Refinery Generators2	03. kesäk. 2016
United Oil Refinery Generators	03. kesäk. 2016
United Oil Plant Standby Generators	03. kesäk. 2016
United Oil Office Portable Generators	03. kesäk. 2016
United Oil Installations3	03. kesäk. 2016
United Oil Installations2	03. kesäk. 2016

Records per page

10

Previous 42 Records - Page 1 / 5 Next

Kuva 11. Datatauluesimerkki

Datataulua voidaan käyttää uudelleen useaan kertaan samalla tai eri sivuilla. Taulun data näyttää erilaiselta riippuen Builderissä syötetyistä arvoista. Taulun avulla voidaan luoda esimerkiksi raportti sivu, jolla näytetään erilaista dataa monesta eri objektista (kuva 12).

Sales ▾ Home ▾ Opportunities ▾ Leads ▾ Tasks ▾ Files ▾ Notes ▾ Accounts ▾ Contacts ▾ Campaigns ▾ Dashboards ▾ Reports ▾ Chatter ▾ Groups ▾ Calendar ▾ People ▾ Cases ▾ Forecasts

---

### Opportunities

NAME ↑	ACCOUNT NAME	AMOUNT	STAGE	CREATED DATE
Burlington Textiles Weaving Plant Generator	Burlington Textiles Corp of America	235000.00	Closed Won	Jun 03, 2016
Chairs to meeting rooms	Burlington Textiles Corp of America	10000000.00	Needs Analysis	Mar 27, 2017
Computers for factories	Burlington Textiles Corp of America	1000000.00	Prospecting	Mar 27, 2017
Express Logistics Portable Truck Generators	Express Logistics and Transport	4000000.00	Value Proposition	Jun 03, 2016
Express Logistics SLA	Express Logistics and Transport	120000.00	Perception Analysis	Jun 03, 2016
Express Logistics Standby Generator	Express Logistics and Transport	220000.00	Closed Won	Jun 03, 2016
for loop 1 Test123	for loop 1	120000.00	Prospecting	Sep 05, 2016
for loop 3 Test123	for loop 3	111111.00	Prospecting	Sep 05, 2016
GenePoint SLA	GenePoint	7500000.00	Closed Won	Jun 03, 2016
Grand Hotels Emergency Generators	Grand Hotels & Resorts Ltd	210000.00	Closed Won	Jun 03, 2016

Records per page: 10

Previous 30 Records - Page 1 / 3 Next

---

### Leads

FULL NAME ↑	EMAIL	ACCOUNT NAME ↑	BILLING CITY
Andy Young	a_young@dickenson.com	asdtst	null
Bertha Boxer	bertha@fcor.net	Burlington Textiles Corp of America	Burlington
Betty Blair	bblair@abankingco.com	Dickenson plc	Lawrence
Bill Dadio Jr	bill_dadio@zenith.com	Digi digi	null
Brenda McClure	brenda@cardinal.net	Edge Communications	Austin
Carolyn Crenshaw	carolync@acelis.com	Express Logistics and Transport	Portland
David Monaco	david@blues.com	for loop 1	null
Eugena Luce	eluce@pacificretail.com	for loop 2	null
Jack Rogers	jrogers@btca.com	for loop 3	null
Jeff Glimpse	jeff@jackson.com	GenePoint	Mountain View

Previous 23 Records - Page 1 / 3 Next

---

### Accounts

FULL NAME ↑	CREATED DATE	ACTIVE
Admin User	Jun 03, 2016	true
Chatter Expert	Jun 03, 2016	true
Gabriel Takel	Jun 03, 2016	true
TakeSite Site Guest User	Aug 01, 2016	true
test Site Guest User	Nov 03, 2016	false
Testtest Site Guest User	Nov 07, 2016	false
Tuomas Kinuski	Jun 28, 2016	true
Tuomio Kinttunen	Jul 13, 2016	true
XYZ Site Guest User	Mar 10, 2017	true

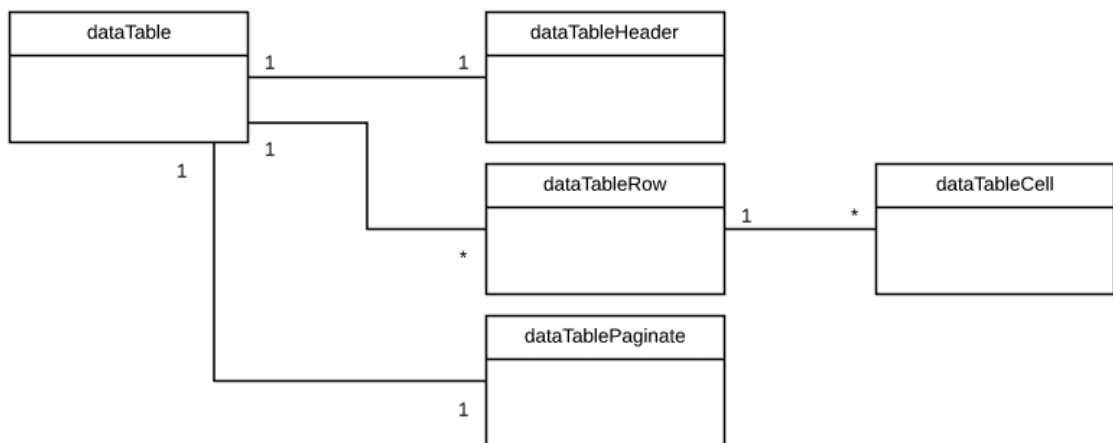
Previous 9 Records - Page 1 / 1 Next

Kuva 12. Esimerkki sivusta, jossa datataulu komponenttia käytetään useaan kertaan

#### 5.4 Datataulukomponentin rakenne ja toiminnallisuus

Datataulukomponentti kokonaisuudessaan sisältää useamman muun komponentin, sekä muutaman tapahtuman (kuva 13):

- `dataTable` – päätason komponentti, joka sisältää muut komponentit.
- `dataTableHeader` – otsikkokomponentti, joka sisältää sarakkeiden otsikot ja joka laukaisee taulukon lajittelu tapahtumia.
- `dataTableRow` – rivikomponentti, joka pitää sisällään yhden tietueen datan ja yhden tai useamman solukomponentin.
- `dataTableCell` – solukomponentti, joka sisältää yhden tietueen yhden kentän datan.
- `dataTablePaginate` – sivunvaihtokomponentti, joka laukaisee sivunvaihtotapahtumia.



Kuva 13. Datataulun komponenttirakenne.

Rakenteeltaan datataulu on hyvin yksinkertainen kokonaisuus, sillä tapahtumien määrä on pieni. Tapahtumia käytetään ainoastaan datan lajittelun vaihtamiseen ja näytettävän sivun vaihtamiseen.

Datataulun alustuksen yhteydessä data haetaan datataululle syötettyjen arvojen mukaisesti palvelimelta. Komponentin Apex-luokka on kirjoitettu dynaamiseksi, eikä se sisällä suoria viittauksia mihinkään erityiseen objektiin tai kenttään (liite 15). Apex-luokka saa parametreina hakuun tarvittavat tiedot suoraan komponentille Builderissa asetetuista arvoista. Apex käsittelee datan ja tekee siitä listan String-tyyppisiä merkkijonoja. Lista palautuu dataTable-komponentille, joka käsittelee merkkijonot ja asettaa taulun otsikot ja datat sen sisäisille komponenteille.

## 5.5 Testaus

Salesforce havaitsee yleensä koodeissa tapahtuvat virheet ja antaa niistä käyttöliittymässä virheilmoituksen. Tässä insinööriyössä tehdyille komponenteille ei tehty varsinaista testaussuunnitelmaa. Kehitysvaiheessa komponentteja testattiin jatkuvasti niiden kehityksen aikana kehittäjän toimesta. Kehityksen aikana tapahtuneella testauksella varmistettiin, että komponentit toimivat oikealla tavalla eivätkä ne aiheuta virheitä.

Salesforce ei aina välttämättä kuitenkaan havaitse kaikkea, esimerkiksi jos kyseessä ei ole virhe koodissa vaan logiikassa. Tällöin virhe tulee esille komponenttia käyttämällä. Korttielementti ja sivunumerointi komponenttien osalta virheitä ei ilmentynyt. Datataulu komponentin osalta muutama pieni virhe ilmeni komponentin käytön aikana. Virheet korjattiin nopeasti ja komponentti uudelleen paketoitiin komponenttikirjastoon.

Datataulukomponentti oli työn ainoa komponentti, jolla oli Apex-kontrolleri. Apex-kontrollereille täytyy tehdä yksikkötestit, mikäli se halutaan tuotantoympäristöön. Yksikkötestien tulee kattaa vähintään 75 % Apex-luokan koodista. Datataulun yksikkötestiluokka (liite 16) kattaa Apex-kontrollerista 100 %.

## 6 Uudelleenkäytettävyyden tuomat hyödyt Fluidossa

Komponenttien valmistumisen jälkeen ne on paketoitu Salesforcea niin sanotuksi hallitsemattomaksi paketiksi. Komponentin pystyy asentamaan mihin tahansa Salesforce-ympäristöön linkin ja salasanan avulla. Hallitsemattoman paketin ansiosta komponentteja voidaan muokata jokaisessa ympäristössä siten, kuten halutaan. Asentaminen ei kestä muutamaa minuuttia kauempaa, ja ne voidaan myös siirtää suoraan tuotantoympäristöihin.

Tämän työn aikana datataulukomponenttia on hyödynnetty useammassa projektissa, ja sen käyttö on todettu helpoksi ja tehokkaaksi järjestelmävalvojen toimesta. Komponentin avulla on säästetty useampi sata työtuntia, koska se on täysin dynaaminen eikä datataulua tarvitse rakentaa alusta asti, mikäli siihen halutaan lisää ominaisuuksia.

Sivunumerointi- ja korttielementtikomponentteja on hyödynnetty muutamassa eri projektissa ja niiden ansiosta kehitykseen mennyttä aikaa on säästetty noin parin työpäivän verran.

Kaikki komponentit ovat Fluidolla sisäisesti komponenttikirjastossa jaossa, josta kaikki kehittäjät ja konsultit voivat asentaa komponentteja eri Salesforce-ympäristöihin. Tämän työn tekijä vastaa mm. tämän työn aikana tehdyistä komponenteista, niiden ylläpidosta, päivittämisestä, dokumentoinnista ja kehittämisestä.

Työn tuloksista havaittiin, että Lightning-komponentteja voidaan käyttää uudelleen monella eri tasolla. Konsultit ja järjestelmävalvojat voivat hyödyntää uudelleenkäytettäviä komponentteja ymmärtämättä koodia. Kehittäjät voivat jatkokehittää niitä tai hyödyntää niitä muissa komponenteissa. Molemmissa tapauksissa havaittiin, että Lightning-komponenttien käyttäminen uudelleen säästää aikaa. Esimerkiksi työssä tehtyä datataulukomponenttia hyödynnettiin useassa projektissa.

Suurimpana haasteena työssä oli tehdä komponenteista täysin dynaamisia ja riippumattomia muista konfiguraatioista. Dynaamisten komponenttien luominen vaatii tarkkaa suunnittelua siitä, kuinka komponenttien tulisi teknisesti toimia. Suunnitteluun vaikutti Lightning-viitekehityksen rajoitteet, koska tämän työn aikana viitekehitys oli vielä uusi ja siinä oli paljon puitteita. Viitekehitys on kehittynyt paljon tämän työn aikana, ja muun muassa työssä tehty datataulukomponentin voisi rakentaa yksinkertaisemmin.

Toinen erittäin haastava tekijä oli komponenttien jakaminen. Vaikka komponentit olivat kirjastossa saatavilla, niin niiden olemassa olon tiedottaminen oli haastavaa. Fluidossa ei aikaisemmin ollut uudelleenkäytettävyyden prosessia, joten käsite yrityksessä oli uusi. Prosessi ei työn aikana vielä ehtinyt täysin käynnistyä, joten siinä on vielä hieman kehitettävää. Myös muiden kehittäjien tekemien komponenttien saaminen kirjastoon tuntui olevan haastavaa.

Tämän työn aikana uudelleenkäytön prosessin määrittely ja käynnistäminen oli vielä kesken. Komponenttikirjasto kuitenkin ehdittiin ottamaan käyttöön työn aikana. Kirjasto toimii käytännössä tietokantana, johon kehittäjät voivat luoda uusia tietueita uudelleenkäytettävistä komponenteista. Tietueet voivat pitää sisällään muun muassa linkit komponenttien asennusosoitteisiin ja lähdekoodiin. Tietueisiin voi myös lisätä komponenttien kuvaukset ja tiedon siitä, kuka komponentin on tehnyt.

## 7 Yhteenveto

Insinööriyössä tutustuttiin uudelleenkäytettävyyteen ja sen ympärillä oleviin asioihin. Uudelleenkäytettävyyteen liittyy paljon huomioon otettavaa, mikäli sitä halutaan toteuttaa toimivasti. Todettiin, että uudelleenkäytettävyyttä varten olisi hyvä olla prosessi.

Salesforce on yksi maailman suurimmista pilvipalvelutuottajista. Salesforce on luonut palveluunsa Lightning-viitekehityksen, joka perustuu komponenttiajattelutapaan. Lightning-komponentit ovat Salesforcen mukaan uudelleenkäytettäviä käyttöliittymän osia.

Työssä pyrittiin selvittämään Lightning-komponenttien uudelleenkäytettävyyttä Fluidon Oy:lle. Työn tuloksena oli useampi Lightning-komponentti, jotka ovat Fluidolla sisäisesti jaossa komponenttikirjastossa. Työn suurimmaksi kehitettäväksi komponentiksi valittiin datataulukomponentti. Datatauluja käytetään useassa paikassa Salesforcessa, mutta Salesforcen omassa taulukomponentissa oli paljon puutteita.

Uudelleenkäytettävien Lightning-komponenttien kehittäminen mahdollistaisi Fluidolle tuotekehityksen tapaista liiketoimintaa. Niiden avulla voidaan tehdä valmiita ratkaisuja eri asiakkaille, ja niiden käyttöönotto olisi erittäin nopeaa. Ne säästäisivät aikaa monessa eri asiassa kuten kehityksessä, testaamisessa ja dokumentoinnissa.

Ideoita uudelleenkäytettävistä Lightning-komponenteista voi olla loputon määrä. Niiden kehittämisestä voisi tehdä useita insinööritöitä. Datataulu voisi mahdollisesti myös jatkokehittää. Työssä tehty datataulu on ainoastaan datan näyttämistä varten, mutta jatkokehityksenä siihen voisi lisätä mahdollisuuden muokata dataa.



## Lähteet

Eloquent JavaScript. Marjin Haverbeke. Verkkodokumentti. <http://eloquentjavascript.net/>. Luettu 18.4.2017.

Journal of Theoretical and Applied Information Technology. Jalender, Govardhan, Premchand. A Pragmatic Approach To Software Reuse. Verkkodokumentti. <https://pdfs.semanticscholar.org/b72a/a8cddb12044d26fbf818bad30bb1fb22d8e4.pdf>. Luettu 23.4.2018.

Lombard Hill Group. What Is Software Reuse?. Verkkodokumentti. [http://lombardhill.com/what\\_reuse.htm](http://lombardhill.com/what_reuse.htm). Luettu 26.2.2017.

Salesforce.com. 2017. Introducing Apex. <https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/>. Luettu 7.4.2017.

Salesforce.com. 2017. Introducing Visualforce. <https://developer.salesforce.com/docs/atlas.en-us.pages.meta/pages/>. Luettu 7.4.2017.

Salesforce.com. 2017. Lightning Components Developer Guide. <https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning>. Luettu 27.3.2017.

Salesforce.com. 2017. Lightning Components Developer Guide Handbook. <https://resources.docs.salesforce.com/sfdc/pdf/lightning.pdf>. Luettu 26.2.2017.

Salesforce.com. 2017. Standard Objects. [https://developer.salesforce.com/docs/atlas.en-us.object\\_reference.meta/object\\_reference/sforce\\_api\\_objects\\_list.htm](https://developer.salesforce.com/docs/atlas.en-us.object_reference.meta/object_reference/sforce_api_objects_list.htm). Luettu 7.4.2017.

Salesforce.com 2017. What is Salesforce?. <https://www.salesforce.com/eu/crm/what-is-salesforce/>. Luettu 27.3.2017.

University of California. Douglas C Schmidt. 2006. Why Software Reuse has Failed and How to Make It Work for You. Verkkodokumentti. <http://www1.cse.wustl.edu/~schmidt/reuse-lessons.html>. Luettu 24.4.2017.

w3.org. 1998. A history of HTML. Dave Raggett. Verkkodokumentti. <https://www.w3.org/People/Raggett/book4/ch02.html>. Luettu 18.4.2017.

w3schools.com. 2017. HTML Introduction. [https://www.w3schools.com/html/html\\_intro.asp](https://www.w3schools.com/html/html_intro.asp). Luettu 18.4.2017.

w3schools.com. 2017. HTML5 Introduction. [https://www.w3schools.com/html/html5\\_intro.asp](https://www.w3schools.com/html/html5_intro.asp). Luettu 18.4.2017.

## dataTable.cmp -resurssi

```

5 <aura:component controller="DataTableController"
6     implements="flexipage:availableForAllPageTypes,forceCommunity:availableForAllPageTypes,force:hasRecordId"
7     access="global">
8
9     <!-- ##### Design attributes ##### -->
10    <aura:attribute name="title" type="String" default="Hello" access="global" />
11    <aura:attribute name="subj" type="String" access="global" default="Opportunity"/>
12    <aura:attribute name="fields" type="String" access="global" default="Name,CreateDate"/>
13    <aura:attribute name="clause" type="String" access="global" default="Amount > 10000"/>
14    <aura:attribute name="sortBy" type="String" access="global" default="Name"/>
15    <aura:attribute name="recPerPage" type="Integer" access="global" default="10"/>
16    <aura:attribute name="showPagePicklist" type="Boolean" access="global" default="true"/>
17    <aura:attribute name="listOfRecPerPageValues" type="String" access="global" default="10,20,30,40,50"/>
18    <aura:attribute name="recordLink" type="Boolean" access="global" default="false"/>
19    <aura:attribute name="recordIdObject" type="String" access="global" />
20    <aura:attribute name="sortOrder" type="String" default="DESC" access="global"/>
21    <!-- ##### Design attributes ##### -->
22
23    <aura:attribute name="recPerPageList" type="Integer[]" />
24    <aura:attribute name="pageList" type="List" />
25    <aura:attribute name="page" type="integer" />
26    <aura:attribute name="pages" type="integer" />
27    <aura:attribute name="total" type="integer" />
28    <aura:attribute name="data" type="String[]" access="global" />
29    <aura:attribute name="fieldAPIS" type="String[]" access="global" />
30    <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
31    <aura:dependency resource="markup://force:showToast" type="EVENT" />
32    <aura:handler event="force:showToast" action="{!c.doInit}" />
33    <aura:handler name="dataTablePageChangeEvent" event="c:dataTablePageChangeEvent" action="{!c.pageChange}" />
34    <aura:handler name="dataTablesSortEvent" event="c:dataTableSortEvent" action="{!c.sortTable}" />
35
36    <aura:if isTrue="{!v.data.length > 0}">
37        <div style="slds-scope">
38            <h1 class="slds-page-header__title">{!v.title}</h1>
39            <table class="slds-table slds-table_bordered slds-table_cell-buffer slds-table_stripped slds-max-medium-table_stacked">
40                <thead>
41                    <tr class="slds-line-height_reset">
42                        <aura:if isTrue="{!v.recordLink}">
43                            <th></th>
44                        </aura:if>
45                        <aura:iteration items="{!v.fieldAPIS}" var="field">
46                            <:dataTableTableHeader field="{!field}" subj="{!v.subj}" sortField="{!v.sortBy}" sortOrder="{!v.sortOrder}" />
47                        </aura:iteration>
48                    </tr>
49                </thead>
50                <tbody>
51                    <aura:iteration items="{!v.data}" var="item">
52                        <tr>
53                            <:dataTableRow data="{!item}" linkIcon="{!v.recordLink}" />
54                        </tr>
55                    </aura:iteration>
56                </tbody>
57            </table>
58            <aura:if isTrue="{!v.showPagePicklist}">
59                <div class="slds-grid slds-grid_align-spread slds-m-top_xx-small">
60                    <lightning:select name="Records per page"
61                        label="Records per page"
62                        value="{!v.recPerPage}"
63                        onchange="{!c.recordsPerPage}"
64                        aura:id="recsPerPage">
65                        <aura:iteration items="{!v.recPerPageList}" var="rec">
66                            <option value="{!rec}">{!rec}</option>
67                        </aura:iteration>
68                    </lightning:select>
69                    <:dataTablePaginate page="{!v.page}" pages="{!v.pages}" total="{!v.total}" />
70                </div>
71            <aura:set attribute="else">
72                <div class="slds-grid slds-grid_align-center">
73                    <:dataTablePaginate page="{!v.page}" pages="{!v.pages}" total="{!v.total}" />
74                </div>
75            </aura:set>
76        </aura:if>
77        <lightning:spinner aura:id="mySpinner"
78            size="small"
79            variant="brand"
80            alternativeText="Loading..."
81            class="slds-hide"/>
82    </div>
83 </aura:if>
84 </aura:component>

```

**dataTableController.js -resurssi**

```
5  doInit : function(component, event, helper){
6      component.set("v.fieldAPIs", component.get("v.fields").split(','));
7      helper.fetchData(component);
8
9
10     if(component.get("v.listOfRecPerPageValues")){
11         var recordsPerPage = component.get("v.listOfRecPerPageValues").split(',');
12         var recordsPerPageInt = [];
13         recordsPerPage.forEach(function(a){
14             recordsPerPageInt.push(parseInt(a,0));
15         });
16
17         component.set("v.recPerPageList", recordsPerPageInt);
18     }
19 },
20
21 pageChange : function(component, event, helper){
22     var direction = event.getParam("direction");
23     if(direction === "previous")
24     {
25         component.set("v.page", (component.get("v.page")-1));
26         component.set("v.data", component.get("v.pageList")[component.get("v.page")-1]);
27     }
28     else if(direction === "next")
29     {
30         component.set("v.page", (component.get("v.page")+1));
31         component.set("v.data", component.get("v.pageList")[component.get("v.page")-1]);
32     }
33 },
34
35 recordsPerPage : function(component, event, helper){
36     component.set("v.recPerPage", component.find("recsPerPage").get("v.value"));
37     helper.fetchData(component);
38 },
39
40 sortTable : function(component, event, helper){
41     component.set("v.sortBy", event.getParam("field"));
42     component.set("v.sortOrder", event.getParam("sortOrder"));
43     helper.fetchData(component);
44 },
45 })
```

## dataTableHelper.js -resurssi

```
4  ({
5  fetchData : function(component, event){
6      var action = component.get("c.getData");
7      action.setParams({"sobj":component.get("v.sObj"),
8                      "fields":component.get("v.fields"),
9                      "clause":component.get("v.clause"),
10                     "sortBy":component.get("v.sortBy"),
11                     "sortOrder":component.get("v.sortOrder"),
12                     "recordId":component.get("v.recordId"),
13                     "recordIdObject":component.get("v.recordIdObject")});
14     action.setStorable();
15     var spinner = component.find("mySpinner");
16     $A.util.removeClass(spinner, "slds-hide");
17     action.setCallback(this, function(a){
18         var state = a.getState();
19         if(state === 'SUCCESS'){
20             var helpTable = a.getReturnValue();
21             var returnTable=[];
22             component.set("v.total", helpTable.length);
23             component.set("v.pages", Math.ceil(helpTable.length/component.get("v.recPerPage")));
24
25             for(var i=0;i<component.get("v.pages");i++)
26             {
27                 var tempTable=[];
28                 for(var j=0;j<component.get("v.recPerPage");j++)
29                 {
30                     if(helpTable.length>0)
31                         tempTable.push(helpTable.shift());
32                 }
33                 returnTable.push(tempTable);
34             }
35
36             component.set("v.pageList", returnTable);
37             component.set("v.data", returnTable[0]);
38             component.set("v.page", 1);
39
40             $A.util.addClass(spinner, "slds-hide");
41         }
42         else if(state === 'ERROR'){
43             var errors = a.getError();
44             if (errors) {
45                 if (errors[0] && errors[0].message) {
46                     $A.log("Error message: " +
47                         errors[0].message);
48                 }
49             } else {
50                 $A.log("Unknown error");
51             }
52         }
53     });
54     $A.enqueueAction(action);
55 }
56 })
```

## dataTable.design -resurssi

```
5 <design:component >
6   <design:attribute name="title" label="Title" description="Title of table" />
7   <design:attribute name="sObj" label="sObject (FROM) *REQUIRED*"
8     description="API name of sObject" />
9   <design:attribute name="fields" label="fields (SELECT) *REQUIRED*"
10    description="Comma separated API name of fields" />
11  <design:attribute name="clause" label="clause (WHERE)"
12    description="SQL WHERE clause, do not include word WHERE in the beginning" />
13  <design:attribute name="sortBy" label="Initial sort by *REQUIRED*"
14    description="Sort by field, give API name" />
15  <design:attribute name="sortOrder" label="Initial sort order"
16    description="Type ASC or DESC" />
17  <design:attribute name="recPerPage" label="Records per page *REQUIRED*"
18    description="How many records will be shown on a page" />
19  <design:attribute name="showPagePicklist" label="Show records per page picklist"
20    description="When checked the tool will show picklist for users
21      to change how many records per page are showed." />
22  <design:attribute name="listOfRecPerPageValues" label="Records per page picklist values"
23    description="Comma separated values eg: 10,20,30,40,50" />
24  <design:attribute name="recordLink" label="Display icon/link to record"
25    description="If checked 1st column will have icon link to the record page" />
26  <design:attribute name="recordIdObject" label="Related object field name"
27    description="API Name of the related object of the record, for example 'AccountId'.
28      Use this field if you want to use the table as a related list.
29      Leave blank if you dont use it.
30      NOTE: only usable on pages where recordId is available." />
31 </design:component>
```

**dataTableTableHeader.cmp -resurssi**

```
1 <aura:component controller="DataTableController">
2   <aura:attribute name="field" type="String" access="global"/>
3   <aura:attribute name="sobj" type="String" access="global"/>
4   <aura:attribute name="sortOrder" type="String" access="global" />
5   <aura:attribute name="sortField" type="String" access="global" />
6   <aura:attribute name="fieldLabel" type="String" access="global"/>
7   <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
8   <aura:handler name="change" value="{!v.sortField}" action="{!c.removeArrow}" />
9   <aura:registerEvent name="dataTableSortEvent" type="c:dataTableSortEvent"/>
10
11 <th scope="col" class="slds-is-sortable slds-text-title_caps" onclick="{!c.sortTable}" >
12   <span style="float:left;" class="slds-truncate"
13     title="{!v.fieldLabel}">{!v.fieldLabel}
14   </span>
15   <span>
16     <lightning:icon class="slds-hide" aura:id="up" iconName="utility:arrowup"
17       size="x-small" alternativeText="ascending order"/>
18     <lightning:icon class="slds-hide" aura:id="down" iconName="utility:arrowdown"
19       size="x-small" alternativeText="descending order"/>
20   </span>
21 </th>
22 </aura:component>
```

## dataTableTableHeaderController.js -resurssi

```

1  ({
2  doInit : function(component, event, helper) {
3      var action = component.get("c.getLabel");
4      action.setParams({"subj":component.get("v.subj"),
5                      "field":component.get("v.field")});
6      action.setCallback(this, function(a){
7          var state = a.getState();
8          if(state === 'SUCCESS'){
9
10             component.set("v.fieldLabel", a.getReturnValue());
11
12             var upArrow = component.find("up");
13             if(component.get("v.field") === component.get("v.sortField")){
14                 $A.util.addClass(upArrow, "slds-show");
15                 $A.util.removeClass(upArrow, "slds-hide");
16             }
17
18         }
19         else if(state === 'ERROR'){
20             var errors = response.getError();
21             if (errors) {
22                 if (errors[0] && errors[0].message) {
23                     $A.log("Error message: " +
24                         errors[0].message);
25                 }
26             } else {
27                 $A.log("Unknown error");
28             }
29         }
30     });
31     $A.enqueueAction(action);
32 },
33
34 sortTable : function(component, event, helper){
35     var sortOrder = component.get("v.sortOrder");
36     sortOrder = sortOrder === 'ASC' ? 'DESC' : 'ASC';
37     component.set("v.sortOrder", sortOrder);
38
39     var sort = component.getEvent("dataTableSortEvent");
40     sort.setParams({ "field": component.get("v.field")});
41     sort.setParams({ "sortOrder": component.get("v.sortOrder")});
42     sort.fire();
43
44     setTimeout(function(){ }, 3000);
45
46     var upArrow = component.find("up");
47     var downArrow = component.find("down");
48     if(component.get("v.sortField") === component.get("v.field")){
49         if(component.get("v.sortOrder") === 'ASC'){
50             $A.util.addClass(upArrow, "slds-show");
51             $A.util.removeClass(upArrow, "slds-hide");
52             $A.util.addClass(downArrow, "slds-hide");
53             $A.util.removeClass(downArrow, "slds-show");
54         }
55         else {
56             $A.util.addClass(downArrow, "slds-show");
57             $A.util.removeClass(downArrow, "slds-hide");
58             $A.util.addClass(upArrow, "slds-hide");
59             $A.util.removeClass(upArrow, "slds-show");
60         }
61     }
62 },
63
64 removeArrow : function(component, event, helper){
65     if(component.get("v.sortField")!== component.get("v.field")){
66         var upArrow = component.find("up");
67         var downArrow = component.find("down");
68         $A.util.addClass(downArrow, "slds-hide");
69         $A.util.removeClass(downArrow, "slds-show");
70         $A.util.addClass(upArrow, "slds-hide");
71         $A.util.removeClass(upArrow, "slds-show");
72     }
73 },
74 })

```

**dataTableRow.cmp -resurssi**

```
1 <aura:component access="global">
2     <aura:attribute name="data" type="String" access="global" />
3     <aura:attribute name="dataset" type="String[]" access="global" />
4     <aura:attribute name="linkIcon" type="Boolean" access="global" />
5     <aura:attribute name="recordId" type="String" access="global" />
6     <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
7
8     <aura:if isTrue="{!v.linkIcon}">
9         <td class="slds-truncate">
10            <a href="{! '/' + v.recordId}" target="_blank">
11                <lightning:icon aura:id="down" iconName="utility:description"
12                    size="small" alternativeText="Go to record page"/>
13            </a>
14        </td>
15    </aura:if>
16    <aura:iteration items="{!v.dataset}" var="d">
17        <c:dataTableCell data="{!d}" recordId="{!v.recordId}" />
18    </aura:iteration>
19 </aura:component>
```



**dataTableRowController.js -resurssi**

```
1 ({
2   doInit : function(component, event, helper) {
3     var data = component.get("v.data").split('@S2@');
4     component.set("v.recordId", data.shift().split('@S1@')[2]);
5     component.set("v.dataset",data);
6   },
7 })
```

**dataTableCell.cmp -resurssi**

```
1 <aura:component access="global">
2   <aura:attribute name="data" type="String" access="global" />
3   <aura:attribute name="field" type="String" access="global" />
4   <aura:attribute name="fieldValue" type="String" access="global" />
5   <aura:attribute name="fieldType" type="String" access="global" />
6   <aura:attribute name="recordId" type="String" access="global" />
7   <aura:attribute name="dateField" type="Date" access="global" />
8   <aura:handler name="init" value="{!this}" action="{!c.doInit}" />
9
10  <td class="slds-truncate">
11    <aura:if isTrue="{!v.fieldType == 'DATE' || v.fieldType == 'DATETIME'}">
12      <div class="slds-truncate" title="{!v.fieldValue}" onclick="{!c.dblclick}">
13        <aura:if isTrue="{!v.fieldValue != 'null'}">
14          <lightning:formattedDateTime value="{!v.dateField}"
15            year="numeric" month="short" day="2-digit"/>
16        <aura:set attribute="else">
17          No date
18        </aura:set>
19      </aura:if>
20    </div>
21    <aura:set attribute="else">
22      <div class="slds-truncate" title="{!v.fieldValue}"
23        onclick="{!c.dblclick}">{!v.fieldValue}</div>
24    </aura:set>
25  </aura:if>
26  </td>
27 </aura:component>
```

**dataTableCellController.js -resurssi**

```
1  ({
2  doInit : function(component, event, helper){
3      var sHelper = component.get("v.data").split('@S1@');
4      component.set("v.field", sHelper[0]);
5      component.set("v.fieldType", sHelper[1]);
6      component.set("v.fieldValue", sHelper[2]);
7
8      if(component.get("v.fieldType")==='DATE'){
9          var d = new Date(component.get("v.fieldValue"));
10         component.set("v.dateField", d);
11     }else if(component.get("v.fieldType")==='DATETIME'){
12         var d = new Date(component.get("v.fieldValue").replace(/-/g, "/"));
13         component.set("v.dateField", d);
14     }
15 },
16
17 dblclick : function(component, event, helper) {
18     //console.log('test '+component.get("v.recordId"));
19 }
20 })
```

**dataTablePaginate.cmp -resurssi**

```
5 <aura:component access="global">
6   <aura:attribute name="page" type="integer"/>
7   <aura:attribute name="pages" type="integer"/>
8   <aura:attribute name="total" type="integer"/>
9   <aura:registerEvent name="dataTablePageChangeEvent"
10     type="c:dataTablePageChangeEvent"/>
11
12 <div class="slds-form-element slds-p-top_small">
13   <lightning:button label="Previous"
14     variant="brand"
15     iconPosition="left"
16     onclick="{!c.previousPage}"
17     disabled="{!v.page == 1}"/>
18
19   &nbsp;{!v.total}&nbsp;Records • Page&nbsp;{!v.page} / {!v.pages} &nbsp;
20
21   <lightning:button label="Next"
22     variant="brand"
23     iconPosition="right"
24     onclick="{!c.nextPage}"
25     disabled="{!v.page >= v.pages}"/>
26 </div>
27 </aura:component>
```

**dataTablePaginateController.js -resurssi**

```
4  ({
5    previousPage : function(component, event, helper)
6    {
7        var changePage = component.getEvent("dataTablePageChangeEvent");
8        changePage.setParams({ "direction": "previous"});
9        changePage.fire();
10   },
11   nextPage : function(component, event, helper)
12   {
13       var changePage = component.getEvent("dataTablePageChangeEvent");
14       changePage.setParams({ "direction": "next"});
15       changePage.fire();
16   }
17 })
```

**dataTablePageChangeEvent -tapahtuma**

```
5 <aura:event type="COMPONENT" description="dataTablePageChangeEvent">  
6     <aura:attribute name="direction" type="String"/>  
7 </aura:event>
```

**dataTableSortEvent -tapahtuma**

```
1 <aura:event type="COMPONENT" description="Event template" >  
2     <aura:attribute name="field" type="String" />  
3     <aura:attribute name="sortOrder" type="String" />  
4 </aura:event>
```

## DataTableController.cls

```

5 public with sharing class DataTableController {
6
7     @AuraEnabled
8     public static List<String> getData(String subj, String fields, String clause, String sortBy,
9         String sortOrder, String recordId, String recordIdObject) {
10
11         String soql = 'SELECT '+fields+' FROM '+subj;
12         if(!String.isBlank(clause)){
13             soql += ' WHERE '+clause;
14             if(!String.isBlank(recordIdObject))
15                 soql += ' AND '+recordIdObject+' = \''+recordId+'\'';
16         }else if(!String.isBlank(recordIdObject)){
17             soql += ' WHERE '+recordIdObject+' = \''+recordId+'\'';
18         }
19
20         fields = 'Id,'+fields;
21
22         Map<String, Schema.SObjectType> schemaMap = Schema.getGlobalDescribe();
23         Map<String, Schema.SObjectField> fieldMap = schemaMap.get(subj).getDescribe().fields.getMap();
24
25         if(!String.isBlank(sortBy) && !sortBy.contains('.')){
26             Schema.SObjectField field = fieldMap.get(sortBy);
27             Boolean dfr = field.getDescribe().isSortable();
28             if(dfr) soql += ' ORDER BY '+sortBy+' '+sortOrder;
29         } else if(sortBy.contains('.')){
30             String[] helper = sortBy.split('\\. ');
31             Map<String, Schema.SObjectField> fieldMap2 = schemaMap.get(helper[0]).getDescribe().fields.getMap();
32             Schema.SObjectField field = fieldMap.get(helper[1]);
33             Boolean dfr = field.getDescribe().isSortable();
34             if(dfr) soql += ' ORDER BY '+sortBy+' '+sortOrder;
35         }
36
37         List<SObject> data = Database.query(soql);
38         List<String> returnString = new List<String>();
39
40         for(SObject so : data){
41             List<String> str = new List<String>();
42             for(String s : fields.split(',')){
43                 if(s.contains('.')){
44                     String[] helper = s.split('\\. ');
45                     try{
46                         str.add(s+'@S10'
47                             +schemaMap.get(helper[0])
48                             .getDescribe().fields.getMap().get(helper[1]).getDescribe().getType()
49                             +'@S10'+so.getSObject(helper[0]).get(helper[1]));
50                     }catch (Exception e){
51                         str.add('No data');
52                     }
53                 }
54                 else str.add(s+'@S10'
55                     +schemaMap.get(subj).getDescribe().fields.getMap().get(s).getDescribe().getType()
56                     +'@S10'+so.get(s));
57             }
58             returnString.add(String.join(str, '@S20'));
59         }
60         return returnString;
61     }
62
63     @AuraEnabled
64     public static String getLabel(String subj, String field){
65         Map<String, Schema.SObjectType> schemaMap = Schema.getGlobalDescribe();
66         if(field.contains('.')){
67             String[] helper = field.split('\\. ');
68             return helper[0]+' '+helper[1];
69         }
70         Schema.SObjectType subjSchema = schemaMap.get(subj);
71         Map<String, Schema.SObjectField> fieldMap = subjSchema.getDescribe().fields.getMap();
72         return fieldMap.get(field).getDescribe().getLabel();
73     }
74 }

```



**DataTableControllerTest.cls**

```
5 @IsTest
6 private class DataTableControllerTest {
7     static testMethod void testDataTable() {
8         Account acc = new Account(Name = 'TestAccount');
9         insert acc;
10
11         Opportunity opp = new Opportunity(Name = 'Test1111',
12                                           Amount = 1000000,
13                                           StageName = 'Prospecting',
14                                           CloseDate = System.today(),
15                                           AccountId = acc.Id);
16         insert opp;
17
18         String subj = 'Opportunity';
19         String fields = 'Name,Amount,StageName,Account.Name';
20         String clause = 'Amount > 10000';
21         String sortBy = 'Name';
22         String sortOrder = 'ASC';
23         String recordId = acc.Id;
24         String recordIdObject = 'AccountId';
25         String fieldForLabel = 'Name';
26
27         Test.startTest();
28         List<String> data = DataTableController.getData(subj, fields, clause,
29                                                       sortBy, sortOrder, recordId, recordIdObject);
30         String returnLabel = DataTableController.getLabel(subj, fieldForLabel);
31         clause = '';
32         sortBy = 'Account.Name';
33         fieldForLabel = 'Account.Name';
34         returnLabel = DataTableController.getLabel(subj, fieldForLabel);
35         data = DataTableController.getData(subj, fields, clause, sortBy,
36                                           sortOrder, recordId, recordIdObject);
37         Test.stopTest();
38     }
39 }
```