Eelis Melto

Käyttöliittymän näkymän tallentaminen PDF-tiedostoksi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tutkinto-ohjelman nimi

Insinöörityö

6.11.2023

Tiivistelmä

Tekijä:	Eelis Melto
Otsikko:	Käyttöliittymän näkymän tallentaminen PDF-tiedostoksi
Sivumäärä:	26 sivua
Aika:	6.11.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Tech Lead Manager Jouni Repo Lehtori Simo Silander

Tässä insinöörityössä tutkitaan, miten web-sovellukseen voidaan lisätä toiminto, joka mahdollistaa käyttäjän tallentaa tai tulostaa käyttöliittymän näkymän PDF-tiedostoksi ilman tarvetta hyödyntää sovelluksen palvelinta. Tämä toiminto kehitettiin Sinch Finland Oy:n Contact Pro -yhteyskeskussovellukseen asiakkaan toiveesta ja julkaistiin vuoden 2022 neljännellä kvartaalilla.

Insinöörityössä esitetään yksityiskohtaisesti toiminnon kehittämisen vaiheet, jotta lukija voi rakentaa vastaavan toiminnon web-sovellukseensa tai verkkosivuilleen soveltamalla työssä esiteltyjen funktioiden koodia ja ymmärtämällä toiminnon elinkaaren.

Avainsanat: Käyttöliittymä, PDF-tiedosto, JavaScript, Tulostaminen

Abstract

Author:	Eelis Melto
Title:	Saving User Interface View as PDF File
Number of Pages:	26 pages
Date:	6 November 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Jouni Repo, Tech Lead Manager
	Simo Silander, Senior Lecturer

This bachelor’s thesis investigates how a feature can be added to the web application that allows the user to save or print the user interface view as a PDF file without the need to use the server of the application. This feature was developed for Sinch Finland Oy's Contact Pro contact center application at the customer's request and was released in the fourth quarter of 2022.

In the thesis, the steps of developing the feature are presented in detail, so that the reader can build a corresponding feature to another web application or website by applying the code of the functions presented here and understanding the life cycle of the feature.

Keywords: User interface, PDF file, JavaScript, Printing

Sisällys

Lyhenteet

MVC: *Model-View-Controller*. *Malli-Näkymä-Ohjain.* Ohelmistoarkkitehtuuri, jossa tyypillisesti sovelluksen käyttöliittymän koodi erotellaan kolmeen osaan malli, näkymä ja ohjain.

WEB: *World Wide Web*. Hajautettu hypertekstijärjestelmä, joka mahdollistaa hypertekstin siirtämisen HTTP-protokollaan sääntöjen mukaisesti.

HTTP: *Hypertext Transfer Protocol.* Protokolla, jota selaimet ja web-palvelimet käyttävät tiedon siirtämiseen.

PDF: *Portable Document Format*. Siirrettävä tiedostomuoto, joka on kehitetty 1990-luvulla Adoben toimesta.

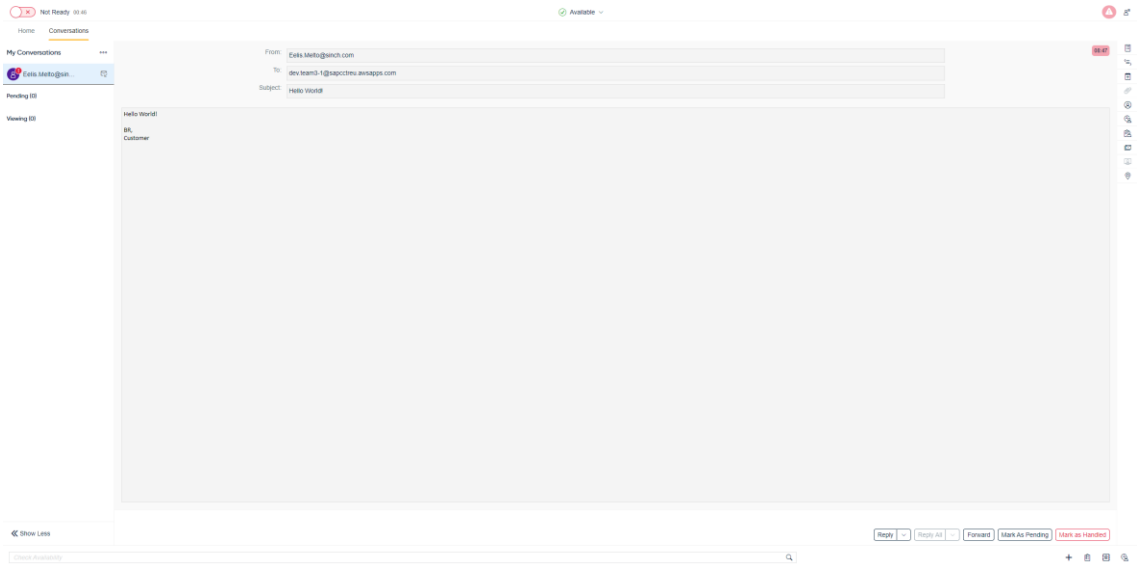
API: *Application Programming Interface*. Ohjelmointirajapinta, joka määrittää, miten eri ohjelmat voivat vaihtaa tietoa keskenään.

MDN: *Mozilla Developer Network.* Dokumentaatiosivusto, joka on tarkoitettu web-ohjelmistokehittäjille ja tarjoaa tietoa ja ohjeita liittyen web-kehitykseen.

1 Johdanto

Insinöörityön tarkoituksena on käydä läpi, kuinka web-sovellukseen kehitetään toiminto, joka mahdollistaa sovelluksen näkymän osion tallentamisen PDF-tiedostoksi niin että PDF-tiedoston luonti ei nojaudu web-sovelluksen palvelimeen.

Työ tehtiin uutena kehityshankkeena Sinch Finland Oy:lle Contact Pro -nimiseen yhteyskeskussovellukseen. Sovelluksen tarpeena oli mahdollistaa tekstiviestien, sähköpostien ja chattien tulostaminen PDF-muotoon sovelluksen käyttäjän toimesta keskustelunäkymältä (kuva 1).



Kuva 1. Contact Pro -yhteyskeskussovelluksen käyttöliittymä, jossa sähköpostiviesti on käsittelyssä keskustelunäkymällä.

Insinöörityössä tullaan käymään läpi toiminnon kehityksen eri vaiheet. Koska työ tehtiin osana kaupallista sovellusta, osa koodista jätetään luottamuksellisuussyistä esittämättä alkuperäisessä muodossaan, mutta kaikki toiminnolle elintärkeä on esitettynä, ja se pohjautuu MDN Web -dokumentaation avoimeen tietoon Web API:n käyttämisestä (1).

2 Suunnittelu

Idea toiminnosta syntyi asiakkaan kehityspyynnöstä. Asiakas toivoi toimintoa, jolla he pystyisivät tallentamaan sähköpostien sisällöt PDF-tiedoston muotoon käyttäjän toimesta. Tämän jälkeen kehitysidea otettiin käsittelyyn ja sen toiminnallisuus päätettiin lisätä tekstiviesti- ja web-chat-keskusteluihin, koska niiden sisältö sopi hyvin tarkoitukseen. Lisäksi PDF-tiedoston sisältö piti olla UX-suunnittelijan tekemän pohjan mukainen ja tiedoston teksti kopioitavissa. Kehitysidea oli kuvan 2 vasemman kulman painike, jota painamalla käyttäjä voisi tallentaa sähköpostien, tekstiviesti- ja web-chat-keskusteluiden sisällön PDF-tiedostoksi. Kun kehitysideasta oli muodostunut selkeä, aloitettiin seuraavaksi sen toteutuksen suunnittelu.

Kuva 2. Sähköpostiviestin painikkeita, jossa PDF-tiedoston tulostava painike on vasemmassa kulmassa reply-painikkeen vieressä.

Toiminnon suunnittelussa selvitettiin, kannattaako toiminto tehdä itse vai otetaanko käyttöön jo valmis kolmannen osapuolen kirjasto, joka hoitaisi PDF-tiedoston luomisen. Lisäksi piti selvittää, pystytäänkö PDF-tiedosto luomaan suoraan käyttäjän koneella selaimen web-rajapinnan avulla niin, että sovelluksen palvelimen koodia ei jouduta muuttamaan sekä millä tavalla toiminto esiintyisi lopulta käyttäjälle.

Käytyämme kollegani kanssa lävitse valmiita avoimenlähdekoodin kirjastoja päädyimme siihen, että teemme toiminnon ilman kolmannen osapuolen kirjastoa. Päätimme sen sijasta hyödyntää selaimen print-metodia (2). Print-metodi on selaimeen sisäänrakennettu toiminto, joka on implementoitu kaikkiin suosituimpiin selaimiin selaimen kehittäjän toimesta. Ideana oli kutsua print-metodia iframe-elementille, koska kirjastoihin liittyi ongelmia, joista vältyttiin, kun toiminto tehtiin täysin omalla toteutustavalla.

2.1 Kolmannen osapuolen kirjastoiden käyttöön liittyvät ongelmat

Valmiita toteutuksia ei voitu ottaa käyttöön, koska ne eivät sopineet hyvin käyttötarkoitukseen. Esimerkiksi PDF.js-kirjaston (3) ongelma oli, että PDF-tiedosto pitäisi olla jo luotuna, että se voidaan ladata käyttäjälle ja sen sisältöä ei voida muokata. Tämä tarkoittaisi sitä, että palvelimen päädyssä täytyisi ensin generoida tietyllä sisällöllä PDF-tiedosto, palauttaa se käyttäjälle ja vasta sitten PDF.js voisi näyttää käyttäjälle PDF-tiedoston sisällön ja tarjota latausmahdollisuuden käyttäjälle.

Toinen ongelma oli, että kolmannen osapuolen kirjaston luoma PDF-tiedosto ei ollut hyvän näköinen ja sen tekstiä ei voinut kopioida tai etsiä. Esimerkiksi jsPDF- (4) ja PDF-LIB-kirjastot (5) eivät soveltuneet tämän takia käyttötarkoitukseen.

Kolmas ongelma, mikä yleisesti liittyy kaikkeen kolmannen osapuolen koodiin, on se, voidaanko olla täysin varma, että kirjasto ei tee mitään muuta, mitä sen ei pitäisi. Etenkin kaupallisessa sovelluksessa, joka käsittelee ihmisten henkilötietoja, täytyy olla varovainen, mitä kolmannen osapuolen kirjastoa käyttää. Sen takia koko kirjaston koodi pitäisi tarkastaa. Mutta tämä on helpommin sanottu kuin tehty, koska koodin tarkastus vie aikaa ja koodi voi olla todella ”kryptistä” ammattilaisellekin. Jos kirjastoa ei ehdi tarkistamaan, on parempi, että sen käyttämisen jättää välistä, jotta voidaan olla aivan varma, että mitään ylimääräistä ei tapahdu taustalla sovelluksen pyöriessä.

2.2 Sopiva kolmannen osapuolen kirjasto

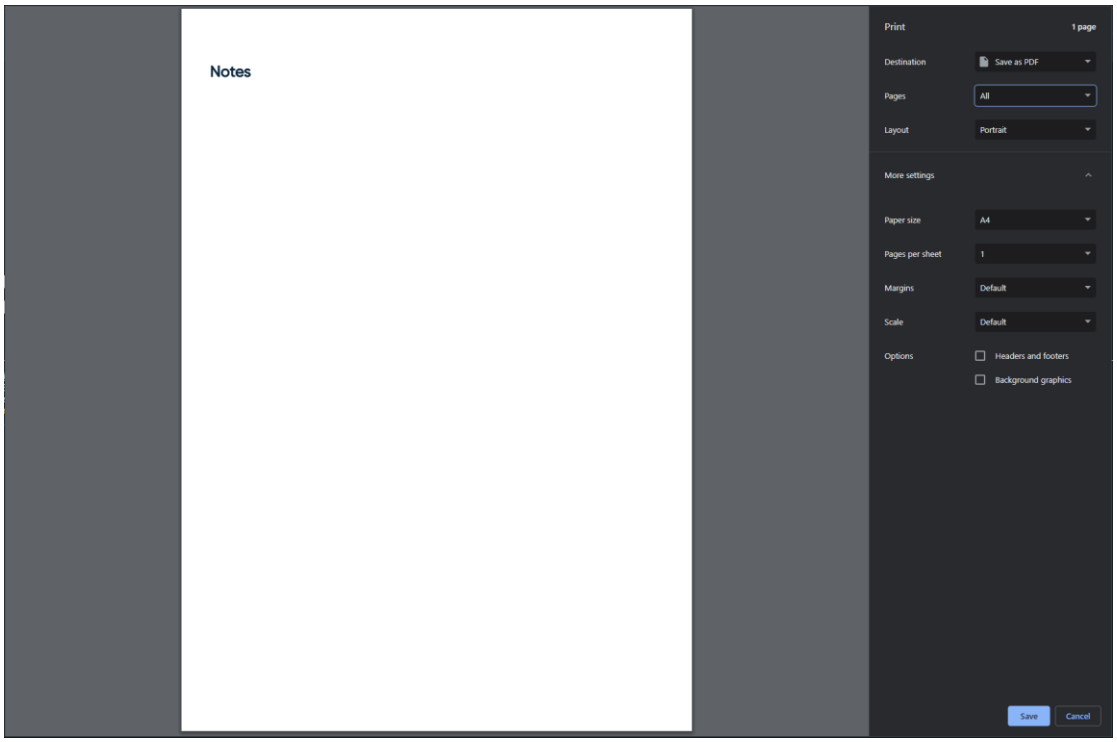
On myös toteutus, joka olisi soveltunut käyttötarkoitukseen esimerkiksi Print.js-kirjasto (6). Tämän kirjaston toteutus on hyvin samankaltainen insinöörityön toteutuksen kanssa, mutta koska se tuli kehityksen loppuvaiheessa vasta vastaan ei sitä päädytty käyttämään. Tämä toki vahvisti ajatusta siitä, että toimintoa oltiin tekemässä käyttötarkoitukseen liittyen oikealla tavalla, koska kirjasto ja työn alla oleva toteutus nojautuivat selaimen tulostusikkunaan (kuva 3).

2.3 Adoben maksullinen rajapinta

PDF-tiedoston kehittäjä Adobe tarjoaa myös maksullista rajapintaa, jonka avulla HTML-merkkijonon voi muuttaa PDF-tiedostoksi (7). Mutta koska rajapinnan käyttö ei ollut ilmaista sen käyttäminen toiminnon tekemiseen poissuljettiin.

2.4 Selaimen web-rajapinnan metodin hyödyntäminen

Suunnittelun alkuvaiheessa syntyi idea selaimen Web-API:n window-rajapinnan (8) print-metodin hyödyntämistä. Tässä vaiheessa emme tienneet, toimiiko metodi käyttötarkoitukseen mutta MDN-dokumentaation antama esimerkki avaamattoman sivun tulostamisesta (9) osoitti, että metodi olisi täydellinen toimintoa varten. Print-metodia käyttämällä selaimen ikkunan dokumentti avautuu selaimen tulostusikkunaan (kuva 3). Tulostusikkunan avulla käyttäjä voi esikatsella PDF-tiedoston sisältöä, muokata sen tyylittelyä, tallentaa tiedoston tai halutessa tulostaa sen valitsemalla tulostimella.



Kuva 3. Selaimen tulostusikkuna.

Tulostusikkunan asetukset ovat

- Destination
- Pages
- Layout
- More settings
- Page size
- Pages per sheet
- Margin
- Scale
- Headers and footers
- Background graphics.

Tulostusikkunan asetusten avulla käyttäjä pystyy tyylittelemään PDF-dokumentin sisällön haluamansa tyyliseksi. Esimerkiksi käyttäjä voi muuttaa Layout-asetuksen landscape-muotoon, jolloin tulostusikkunan sisältö on leveämpi.

Sähköpostiviestien tyylit saattavat vaihdella paljon riippuen sähköpostien käsittelysovelluksesta. Esimerkiksi Outlook lisää lähetettyihin sähköpostiviesteihin erillisiä tyylejä. Tämä saattaa aiheuttaa sen, että sähköpostin sisältöön toimivat tyylit eivät istu oikein, eikä käyttäjä halua käyttää niitä, jotta sisältö olisi PDF-tiedostona helpommin luettava. Tai käyttäjä tulostaa tiedoston tulostimella eikä tulostin tue värillistä tulostamista. Näissä tilanteessa käyttäjä voi ”Background graphics” -asetuksella poistaa kokonaan tyylit käytöstä ja tallentaa sähköpostin sisällön PDF-tiedostoksi, joka ei säilytä alkuperäisen sähköpostin tyylejä.

Print-metodi on HTML-standardin määritelmä (10) ja suosituimmat selaimet kuten Chrome, Edge, Firefox, Opera ja Safari noudattavat tätä standardia. Metodi siis soveltui täydellisesti käyttötarkoitukseen, koska se ei vaadi sovelluksen palvelimen koodiin muutoksia, PDF-tiedoston generointi suoritetaan täysin käyttöliittymän puolella ja sovelluksen tukemat selaimet ovat käyttökelpoisia metodille.

Suoraan kuitenkaan tätä metodia ei voinut käyttää, koska koko sovelluksen dokumentin sisältöä ei haluttu tulostaa, pelkästään tietty osa sovelluksen

dokumentista. Tämä onneksi selvisi MDN-dokumentaation esimerkillä (9). Esimerkin mukaan print-metodia voidaan kutsua iframe-elementille, koska elementti on sisennetty selainkonteksti. Tällöin vain sen sisältö tulostuu ja näkyy tulostusvalintaikkunassa.

Haittapuoli, joka print-metodiin sisältyy, on se, että sen toiminta pysäyttää JavaScriptin säikeen, kunnes tulostusikkuna suljetaan. Tämä ei osoittautunut suureksi ongelmaksi, koska säie kuitenkin suorittaa tulostusikkunan suljettua kaikki prosessit, jotka siltä jäi tulostusikkunan ollessa auki tekemättä.

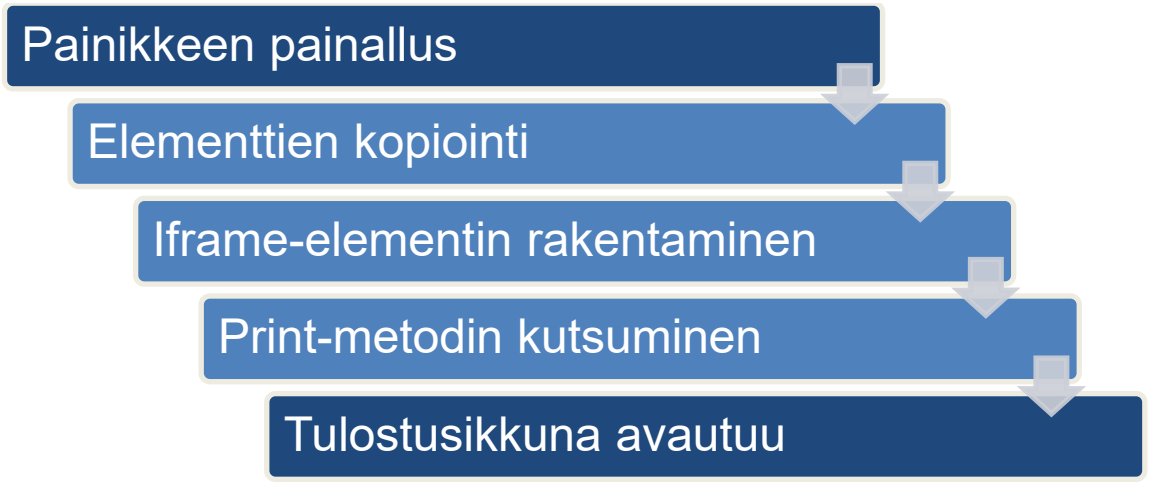
2.5 Toiminnon ilmaantuminen käyttäjälle

Ideana oli, että käyttäjälle toiminto esiintyisi käyttöliittymällä uutena painikkeena sähköposti-, tekstiviesti- ja web-chat-keskusteluiden yhteydessä. Tätä painiketta painamalla selaimen tulostusikkuna avautuisi näkymälle ja esikatseltavan PDF-tiedoston sisältö olisi sama kuin käyttöliittymällä näkyvän keskustelun sisältö.

Sähköposteissa sisällön tyylien säilyttäminen PDF-tiedostossa ei ollut oleellista, joten päädyimme siihen, että tärkeintä on, että teksti on luettavissa ja sen sisältö on esillä. Tekstiviesti- ja web-chat-keskusteluissa halusimme säilyttää keskustelun tyylit ja puhekuplat, joten tavoite oli, että näissä keskusteluissa PDF-tiedosto säilyttäisi käyttöliittymällä olevan tyylin.

3 Toteutus

Suunnittelun jälkeen alkoi toiminnon toteutus. Toteutus voidaan jakaa kolmeen osaan, HTML elementtien kopiointi käyttöliittymältä, iframe-elementin sisällön rakentaminen kopioiduilla elementeillä ja print-metodin kutsuminen iframe-elementin ladattua (kuva 4).



Kuva 4. Prosessikaavio toiminnosta ylhäältä alas. Tummalla merkatut kohdat prosessista ovat alku ja loppu, jotka ilmaantuvat käyttäjälle. Keskikohdat prosessista tapahtuvat taustalla eivätkä näy käyttäjälle.

Sovelluksen kehyksenä toimii SAP UI5, joka on lisenssiä vaativa web-sovelluksien käyttöliittymän rakentamiseen käytetty kehys. Kehyksestä on saatavilla avoimenlähdekoodin versio Open UI5. Insinöörityössä esitetyt kehyksen esimerkkikoodit pohjautuvat Open UI5 -versioon.

UI5 perustuu MVC-arkkitehtuuriin (11), ja toiminto noudattaa tätä arkkitehtuurityyliä. Toimintoa varten luotiin uusi näkymä, johon määriteltiin UI5-painike (12). Esimerkkikoodissa 1 on määritelty XML-kielellä kehyksen mukainen näkymä (13), joka on samankaltainen kuin toimintoa varten luotu uusi näkymä. Näkymässä tärkeintä on määrittää ”controllerName”-ominaisuudelle merkkijono, jossa kerrotaan ohjaimelle määritelty nimi. Kun nimi on määritelty, UI5 saa tiedon siitä, mitä ohjainta näkymän pitää käyttää.

```
<mvc:View controllerName="com.myorg.myapp.controller.PrintButton"
xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc">
      <Button text="Press Me!" press="onPress"/>
</mvc:View>
```

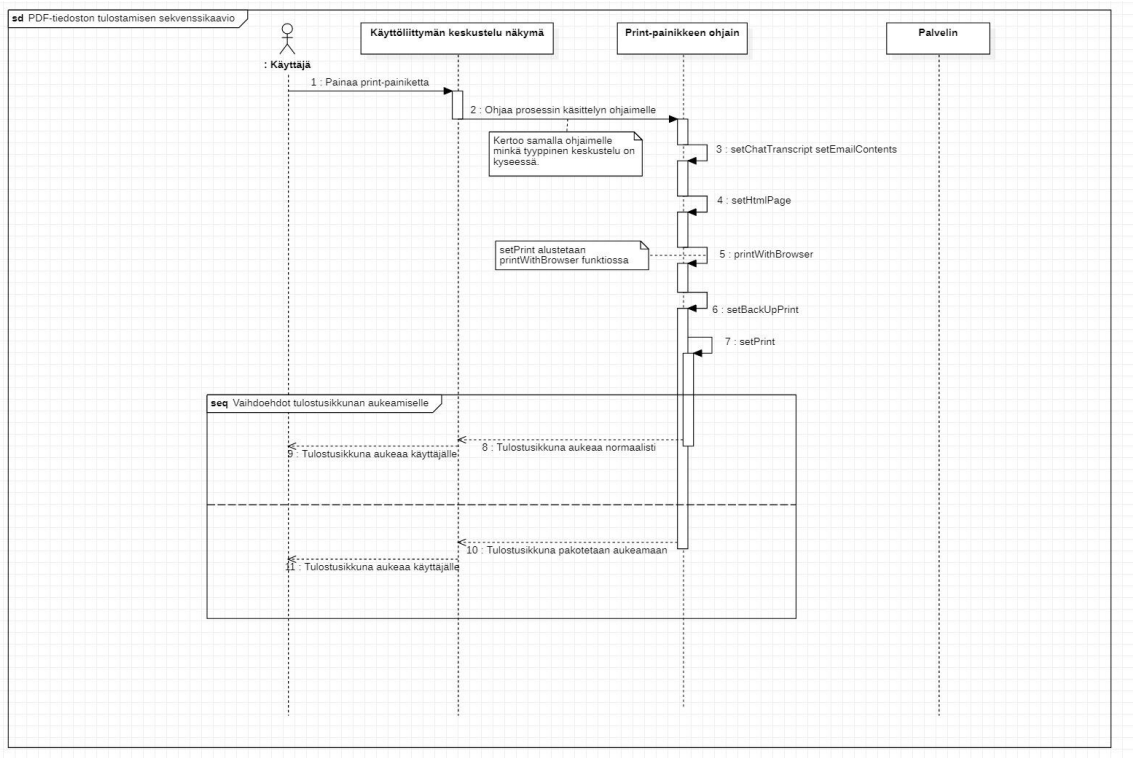
Esimerkkikoodi 1. Open UI5 XML -näkymä esimerkki, johon on määritelty paneeli. Paneelin sisään on laitettu kuva ja painike.

Uuden näkymän luonnin jälkeen tehtiin näkymälle ohjain. Ohjain vastaa siitä, mitä funktioita näkymä voi käyttää. Esimerkkikoodissa 2 on esitettynä ohjain esimerkkikoodin 1 näkymälle. Ohjaimessa on määritelty onPress-funktio. Esimerkkikoodissa 1 on määritelty painike, jolle on annettu painikkeen press-tapahtumakäsittelijäksi onPress-funktio. Eli kun painiketta painetaan, suoritetaan ohjaimessa määritellyn onPress-funktion koodi ja selaimen konsoliin tulostuu ”Button Pressed” -merkkijono.

```
sap.ui.define([
'sap/ui/core/mvc/Controller'
], function(Controller) {
     "use strict";

     var ButtonController = Controller.extend("com.myorg.myapp.control-
     ler.PrintButton",   {

          onPress: function (evt) {
               console.log(“Button Pressed”);
          }
     });
     return ButtonController;
});
```

Esimerkkikoodi 2. Open UI5 -ohjain. Ohjain on määritelty esimerkkikoodin 1 näkymälle ja siihen on määritetty onPress-funktio, joka suoritettaan painikkeen painalluksella.

Kuvan 5 sekvenssikaavio kuvaa toiminnon funktioiden kutsumisjärjestyksen. Kaaviossa on kuvattuna oikealta vasemmalle käyttäjä, näkymä, ohjain ja palvelin. Kun käyttäjä painaa painiketta, käyttöliittymä siirtää painalluksen käsittelyn painikkeen ohjaimelle ja samalla kertoo, mitä keskustelua ollaan tulostamassa. Ohjain kutsuu sisäisiä funktioitaan yksitellen ja lopulta print-metodia päädytään kutsumaan. Palvelin on myös kuvattuna kaaviossa, mutta sille ei lähetetä ollenkaan viestiä. Toiminto suoritetaan siis täysin käyttäjän tietokoneella käyttöliittymän JavaScript-koodilla.



Kuva 5. Sekvenssikaavio tulostusikkunan aukeamisesta. Kaaviossa kuvattuna vasemmalta oikealle käyttäjä, näkymä, ohjain ja palvelin.

3.1 HTML-elementtien kopioiminen käyttöliittymältä

Koska iframe-elementti on luodessa tyhjä, täytyy sen sisältö täydentää käyttöliittymän elementeillä, jotka halutaan näyttää PDF-tiedostossa. Tämän voi halutessa tehdä monella eri tavalla. Tässä tapauksessa tekstiviestien- ja web-chat-keskusteluiden sisältö on kopioitu samalla tavalla ja sähköpostien sisältö suoraan sähköpostieditorin muuttujalta.

3.1.1 Tekstiviesti- ja web-chat-keskusteluiden kopiointi

Tekstiviesti- ja web-chat-keskusteluiden sisältö kopioidaan käyttämällä jQuery-kirjastoa, joka on sisäänrakennettu ohjelmointikehykseen. Esimerkkikoodissa 3 on esillä elementtien kopiointimetodi. Funktio kopioi tietyllä ID:llä olevan elementin lapsineen jQueryn avulla, laittaa elementin html-sisällön div-elementin sisään merkkijonossa ja palauttaa lopulta merkkijonon.

```
_setChatTranscript: function (eId, enabled) {
     var bodyContent = '';
     if (enabled) {
     bodyContent = '<div><br/><h2>'+
     this.getTranslation("ECF_XFLD_CP_PRINT_CHAT_TRANSCRIPT")+'</h2>'
     +jQuery("#" + this.oParentView.byId(eId).getId()).html()+'</div>';
     }
     return bodyContent;
},
```

Esimerkkikoodi 3. Funktio, joka kopioi chat-keskusteluiden sisällön jQuerylla.

Koska kopioitavien elementtien ID:t on määritelty, jQuerya voidaan käyttää etsimään elementit tietyllä ID:llä. Tässä tapauksessa tarvitaan vielä avuksi keskustelunäkymän sisällä pitävä muuttuja oParentView. Muuttuja oParentView on painikkeelle erikseen annettu, jotta ID:n selville saaminen olisi helpompaa. Yleisesti ID:llä elementtien etsiminen ei ole suotavaa, koska elementeillä voi olla automaattinen ID-generointi päällä. Tämä tarkoittaa sitä, että ID:tä ei ole erikseen määritelty vaan ohjelmointikehys luo nämä automaattisesti.

Kuten aikaisemmin mainittiin elementtien kopioinnin voi tehdä monella tapaa. Toinen tapa tehdä tämä sama ilman jQuerya on käyttää JavaScriptin tarjoamaa metodia querySelector (esimerkkikoodi 4). Tämän metodin avulla elementit voidaan hakea CSS-luokan, ID:n tai kompleksimman haun perusteella.

```
document.querySelector("#"+this.oParentView.byId(eId).getId())
```

Esimerkkikoodi 4. querySelector-metodin käyttäminen elementtien kopiointiin jQueryn sijasta.

Täytyy kuitenkin muistaa, että querySelector palauttaa elementin. Elementtiä ei voida suoraan laittaa merkkijonoon vaan tarvitaan elementin html-sisältö merkkijonona. Esimerkkikoodissa 3 elementin html-sisältöön päästiin käsiksi jQuery html-metodin avulla. Tässä tapauksessa html-sisältöön päästään käsiksi elementin innerHTML-ominaisuuden avulla (esimerkkikoodi 5).

```
document.querySelector(“.myClass”).innerHTML
```

Esimerkkikoodi 5. querySelector-metodi hakee elementin, jolla on CSS-luokka ”myClass” ja ominaisuutta innerHTML käytetään html-sisällön saamiseen merkkijonona.

3.1.2 Sähköpostikeskustelun kopiointi

Sähköpostilla voi olla monta tilaa sovelluksessa. Kaikki tilat eivät käytä sähköpostieditoria esimerkiksi sähköpostin pelkkä katsominen historiasta. Jos sähköpostikeskustelu on käsittelyssä, sisältö voidaan kopioida esimerkkikoodin 6 tavalla. Sisältö otetaan talteen muuttujalta, joka pitää sisällään sähköpostieditorin tekstisisältöä.

```
emailContents = this.oRichTextEditor.getValue();
```

Esimerkkikoodi 6. Koodirivi sähköpostiedtorin muuttujan sisällön kopioimisesta.

Sähköposteissa lähettäjän, saajan ja otsikon arvot tallentuvat käyttöliittymän ohjelmistokehyksen SAP UI5:n malliin sähköpostin käsittelyn aloittamisessa. Näitä tietoja sähköpostieditori ei pidä sisällään, joten ne haettiin mallista ja laitettiin p-elementtien sisään. P-elementit puolestaan laitettiin div-elementin sisään, jolla on oma ainutlaatuinen CSS-luokka. Luokan avulla lähettäjä, saaja ja otsikko voidaan tyylitellä erikseen. Jos sähköposti ei ole aktiivinen luetaan sähköpostin sisältö mallista.

```
bodyContent = '<div class="cpPrintEmailLabels"><br/>'
          +'<h2>'+heading+'</h2>'
          +'<p>'+fromAddress+'</p>'
          +'<p>'+toAddress+'</p>'
          +'<p>'+subject+'</p>'
          +'<p>'+attachments+'</p></div>'
          +'<div class="cpPrintEmail"><br/>'+emailContents+'</div>';
```

Esimerkkikoodi 7. Palautuva merkkijono sähköpostikeskusteluista, kun sisältö on kopioitu.

3.2 Iframe-elementin sisällön rakentaminen

Kun sisältö on kopioitu, luodaan esimerkkikoodin 8 funktiolla merkkijono, joka sisältää html-syntaksilla dokumentin rakenteen. Jotta kopioitu sisältö olisi näkyvissä PDF-tiedostossa, on tärkeä laittaa kopioitu sisältö bodyContent-muuttuja body-elementin sisään.

```
_setHtmlPage: function (userProperties,notes,bodyContent) {

// Get pdf document name from model
var title = this.getAppModelProperty("pdfName");

// Add all header content together.
var hContent = '<html lang=""><head>' +
'<title>'+title+'</title>' +
'<link rel="preload" href="css/fonts/Gilroy/Gilroy-Medium.otf"'+
'as="font" type="font/otf" crossorigin>' +
'<link rel="preload" href="css/print.css" as="style">' +
'<link rel="stylesheet" media="print" type="text/css"'+
'href="css/print.css">' +
'<style>li {list-style: none;}</style>' +
'</head><body><div>'+userProperties+notes+'</div>';

// Add header, body and close contents together.
return hContent+"<div>"+bodyContent+"</div></body></html>";

},
```

Esimerkkikoodi 8. Funktio, joka rakentaa kopioiduista elementeistä html-dokumentin merkkijonona.

Link-elementeillä määritellään dokumentin tekstin fontti ja elementtien tyylit. Elementtien tyylit tulevat print.css-nimisestä CSS-tiedostosta. Tätä tiedostoa käytetään vain tulostamiseen, ja se sisältää kaikki tarvittavat samat säännöt kuin sovelluksen pää-CSS-tiedosto. Link-elementin ominaisuus ”rel” on määritelty arvolla ”preload”. Tämän avulla tyylit määritellään tärkeäksi dokumentille ja niiden lataaminen priorisoidaan ennen dokumentin renderöintiä. Tällä saadaan parannettua tulostusikkunan avautumisen nopeutta.

3.3 Print-metodin kutsuminen

Kun sisältö on rakennettu ja valmis annettavaksi iframe-elementille, kutsutaan esimerkkikoodin 9 funktiota. Funktio saa parametriksi merkkijonon, jossa on

määriteltynä html-syntaksilla dokumentti, jonka body-elementti sisältää kopioidun keskustelun elementit. Funktio luo ensimmäisellä rivillä iframe-elementin, toisella rivillä esimerkkikoodin 10 _setPrint-funktio alustetaan iframe-elementin lataus tapahtumaksi. Kolmannella rivillä alustetaan iframe-elementin virhekäsittelyyn käytettävä funktio. Jos virhe tapahtuu, kirjataan sovelluksen LOG-tiedostoon virheviesti debug-tasolle ja käyttäjälle näytetään virheilmoitus. Virheen käsittelijän määrittämisen jälkeen iframe-elementin ominaisuudet määritellään.

```
_printWithBrowser: function (htmlpage, oButton) {
     var iframe = document.createElement('iframe');
     iframe.onload = this._setPrint;
     iframe.onerror = function (event) {
          if (typeof window.chrome === "object") {
                    sap.ecf.communicationpanel.util.Utils.get-
               Instance().setBrowserTabTitleAndFavicon();
          }

          // Log the error
          var oSession = sap.ecf.core.getSession();
          var oLogger = oSession.getLogger();
          oLogger.error("_printWithBrowser - " + event.type + ": "+
          event.message);

          // Show toast message
          MessageToast.show(this.getTranslation("ECF_XMSG_CP_PRINT”+
               ”_ERROR"));
          oSession.getLogger().debug("MessageToast [EN]:”+
               Printing request failed. Try again.");

          // Set Button back to enabled
          oButton.setEnabled(true);
     };
     iframe.id = "printFrame";
     iframe.title = "PDF Document";
     iframe.style.visibility = "hidden";
     iframe.style.display = "contents";
     iframe.loading = "eager";
     iframe.sandbox = "allow-modals allow-same-origin";

     // Add html content to iframe.
     iframe.srcdoc = htmlpage;
     document.body.appendChild(iframe);
     this._setBackUpPrint(iframe,oButton);
},
```

Esimerkkikoodi 9. Iframe-elementin luova funktio, joka saa parametrina html-dokumentin sisällön.

Oleellisimmat ominaisuudet ovat ”style.visibility”, ”sandbox” ja ”srcdoc”. ”Style.visibility” määrittää, onko iframe-elementti näkyvänä käyttäjälle.

”Sandbox” määrittää iframe-elementin rajoitukset. ”Sandbox”-rajoitukset toimivat niin, että kaikki rajoitukset ovat oletuksena päällä iframe-elementille. Lopuksi ominaisuudelle ”srcdoc” annetaan funktiolle parametrina saatu merkkijono ja iframe-elementti sisennetään sovelluksen dokumentin body-osioon, mikä aloittaa iframe-elementin sisällön lataamisen. Viimeinen rivi funktiossa asettaa varmuuskopioinnin päälle, mikä pyrkii estämään tulostusikkunan loputtoman latauksen.

```
_setPrint: function () {
     var oSession = sap.ecf.core.getSession();
     var oLogger = oSession.getLogger();
     oLogger.info("Print window content loaded starting print.");
     this.contentWindow.onbeforeprint = function () {

          // For some reason MessageToast   have close method.
          // So we need to close it this way.
          for (var index in MessageToast._aPopups) {
                    if (MessageToast._aPopups[index]) {
                         MessageToast._aPopups[index].close();
                    }
               }
          };
          this.contentWindow.onafterprint = function (event) {
               this.frameElement.remove();

               // In firefox both message toast are fired on the same
               time.
               // With this if we can select to only show second mes-
               sage in chromium based browsers.
               if (typeof window.chrome === "object") {
               sap.ecf.communicationpanel.util.Utils.getInstance().set-
               BrowserTabTitleAndFavicon();

               // Show toast message
               MessageToast.show(oSession.getResourceProp-
               erty("ECF_XMSG_CP_PRINT_AFTER"));
               sap.ecf.core.getSession().getLogger().debug("Mes-
               sageToast [EN]: Closing print menu.");
               }
          };
     this.contentWindow.focus();
     this.contentWindow.print();
     oLogger.info("Printing function called.");
},
```

Esimerkkikoodi 10. Funktio, joka kutsuu print-metodia iframe-elementille.

Tulostusikkunan avautuminen tapahtuu esimerkkikoodin 10 funktion avulla. Koska _setPrint-funktio on annettu iframe-elementin latauskäsittelijäksi, suoritetaan funktion koodi, kun iframe-elementti on ladannut sisältönsä. Funktiossa

määritellään iframe-elementin contentWindow-ominaisuudelle tapahtumankäsittelijät ennen tulostusta ”onbeforprint” ja tulostuksen jälkeen ”onafterprint”. Tapahtumankäsittelijä ”onbeforeprint” tapahtuu print-metodin kutsumisen jälkeen ennen kuin tulostusikkuna on auennut käyttäjälle. Kun tulostusikkuna sulkeutuu, tapahtumankäsittelijä ”onafterprint” suoritetaan. Toimintoa varten ”onbeforeprint”-tapahtumankäsittelijä tekee kehykseen liittyvän korjauksen, ja ”onafterprint” poistaa iframe-elementin dokumentin rakenteesta. Jos iframe-elementtiä ei poisteta dokumentin rakenteesta se jää ”kummittelemaan” rakenteeseen, kunnes sovellus uudelleen ladataan.

3.3.1 Tulostus toisella selaimen ikkunalla

Mikäli sovelluksen kannalta JavaScript-säikeen pysähtyminen halutaan välttää, voidaan esimerkkikoodin 9 _printWithBrowser-funktion loppuosaa muuttaa. Esimerkkikoodissa 11 on kuvattuna muutos, jonka avulla funktio avaa iframe-elementin toisella ikkunalla selaimessa. Tämän avulla sovellusta pyörittävän ikkunan JavaScript-säie ei pysähdy.

```
var win = window.open();
win.document.body.appendChild(iframe);
```

Esimerkkikoodi 11. Mikäli iframe-elementti halutaan avata uudelle sivulla, täytyy esimerkkikoodin 9 funktiossa luoda uusi ikkuna ja sisentää iframe-elementti uuteen ikkunaan.

3.3.2 Varmuuskopiointi

Varmuuskopioinnin tarkoitus on varmistaa, että tulostusikkuna aukeaa viimeistään 10 sekunnin jälkeen painikkeen painamisesta. Tämä osoittautui tarpeelliseksi keskusteluihin, joihin on liitetty kuvia. Kuvien lataaminen voi kestää kauan etenkin hitaissa verkkoyhteyksissä. Tulostusikkuna ei puolestaan aukea ennen kuin kaikki iframe-elementin resurssit on ladattu. Tämän tilanteen tapahtuminen täytyi estää, koska käyttäjä voi navigoida tulostusikkunan avautumisen aikana toiseen näkymään ja toisessa näkymässä tulostusikkunan aukeaminen saattaisi hämmentää käyttäjää. Pahimmassa tapauksessa resurssien lataus

kestää useita minuutteja, ja tulostusikkuna aukeaisi käyttäjälle vasta usean minuutin jälkeen. Käyttäjä olettaa, että asiat tapahtuvat sekuntien aikana, joten oli parempi luopua kuvien näkymisestä PDF-tiedostossa tässä tapauksessa, koska huonompi tilanne olisi, että aikaa kuluisi useampi minuutti. Esimerkkikoodissa 12 on varmuuskopiointifunktio. Funktio luo ajastimen ja kuuntelijan iframe-elementin load-tapahtumalle. Ajastin pysäyttää iframe-elementin resurssien lataamisen stop-metodilla ja tekee samat operaatiot kuin esimerkkikoodi 10. Kuuntelija poistaa ajastimen, mikäli iframe-elementin lataus onnistui 10 sekunnissa.

```
_setBackUpPrint: function (_iframe, oButton) {

     // Show toast message
     var oSession = sap.ecf.core.getSession();
     MessageToast.show(oSession.getResourceProp-
     erty("ECF_XMSG_CP_PRINT_BEFORE"), {duration: 10000});
     oSession.getLogger().debug("MessageToast [EN]: Opening print
     menu.");

     var iframe = _iframe;
     var myTimeout = setTimeout(function () {
          iframe.contentWindow.stop();
          iframe.contentWindow.onbeforeprint = function () {
               // For some reason MessageToast doesn't have close
               method. So we need to close it this way.
               for (var index in MessageToast._aPopups) {
                    if (MessageToast._aPopups[index]) {
                         MessageToast._aPopups[index].close();
                    }
               }
          };
          iframe.contentWindow.onafterprint = function () {
               this.frameElement.remove();

               // In firefox both message toast are fired on the same
               time.
               // With this if we can select to only show second mes-
               sage in chromium based browsers.
               if (typeof window.chrome === "object") {
                    sap.ecf.communicationpanel.util.Utils.get-
                    Instance().setBrowserTabTitleAndFavicon();

                    // Show toast message
                    MessageToast.show(oSession.getResourceProp-
                    erty("ECF_XMSG_CP_PRINT_AFTER"));
                    oSession.getLogger().debug("MessageToast [EN]:
                    Closing print menu.");
               }
               oButton.setEnabled(true);
          };
          iframe.contentWindow.focus();
          iframe.contentWindow.print();
     },10000);
     // Event listener that listens if iframe.onload function is ini-
     tated.
     iframe.addEventListener('load', function () {
          clearTimeout(myTimeout);
          oButton.setEnabled(true);
     });
},
```

Esimerkkikoodi 12. Varmuuskopiointifunktio. Funktio aloittaa ajastimen 10 sekunnin viiveellä. Kun ajastin laukeaa iframe-elementin resurssien latautuminen, lopetetaan ja print-metodia kutsutaan.

4 Testaus

Toiminnon koodin testaamiseen käytettiin UI5-kehyksen sisäänrakennettua QUnit-kehystä. QUnit-kehys on luotu JavaScript-koodin testaamista varten ja sitä käytetään yleisesti JQuery-kirjastoa käyttävien sovellusten testaamiseen. Esimekkikoodissa 13 on QUnit-testi, joka testaa esimerkkikoodin 8 funktion. Testissä equal-metodin ensimmäiseksi parametriksi annetaan oletettu sisältö, toinen parametri on testattavasta funktiosta syntyvä sisältö ja kolmas parametri merkkijono, joka näytetään testin onnistuessa. QUnit-testeissä yleisesti luodaan mock-data, jolla simuloidaan toimintoa. Esimerkkikoodin 13 mock-datana toimii equal-metodin ensimmäinen parametri.

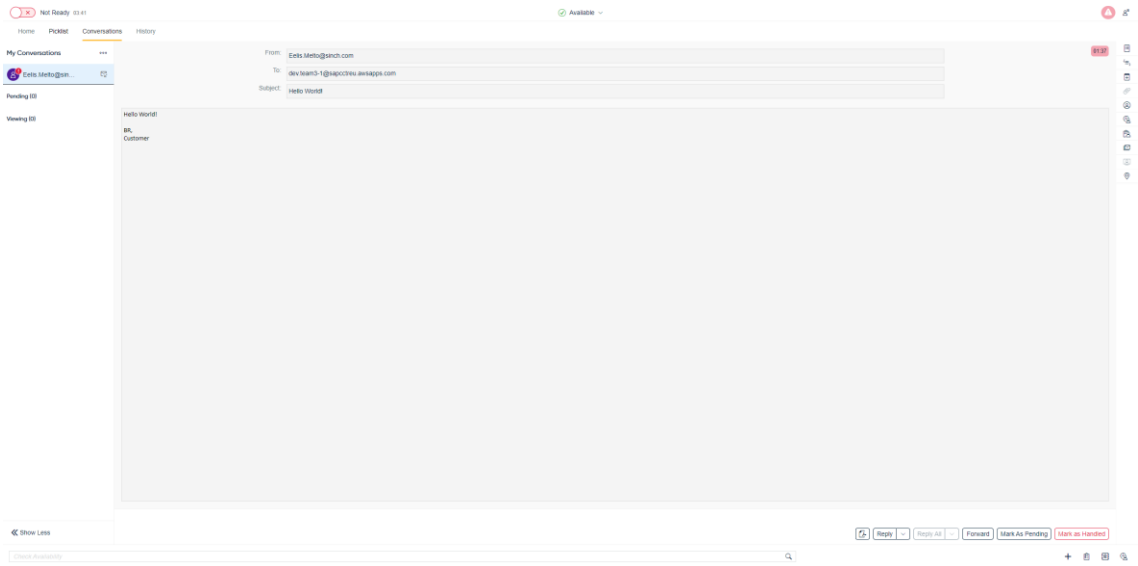
```
QUnit.test("Test _setHtmlPage() event", function (assert) {
          var bResult = (typeof this.oPrintButtonControl-
ler._setHtmlPage === "function");
          assert.ok(bResult, "Event supported");

          assert.equal('<html lang=""><head><title>PDF Document</ti-
tle><link rel="preload" href="css/fonts/Gilroy/Gilroy-Medium.otf"
as="font" type="font/otf" crossorigin><link rel="preload"
href="css/print.css" as="style"><link rel="stylesheet" media="print"
type="text/css" href="css/print.css"><style>li {list-style:
none;}</style></head><body><div></div><div></div></body></html>',
              this.oPrintButtonController._setHtmlPage('','',''),
'Correct value returned.');

          assert.equal('<html lang=""><head><title>PDF Document</ti-
tle><link rel="preload" href="css/fonts/Gilroy/Gilroy-Medium.otf"
as="font" type="font/otf" crossorigin><link rel="preload"
href="css/print.css" as="style"><link rel="stylesheet" media="print"
type="text/css" href="css/print.css"><style>li {list-style:
none;}</style></head><body><div><h2>Conversation De-
tails</h2><h2>Notes</h2></div><div><h2>Tran-
script</h2></div></body></html>',
              this.oPrintButtonController._setHtmlPage('<h2>Conversa-
tion Details</h2>','<h2>Notes</h2>','<h2>Transcript</h2>'), "Correct
value returned.");
     });
```

Esimerkkikoodi 13. QUnit-testi esimerkkikoodin 6 _setHtmlPage-funktiolle.

Jokainen uusi toiminto testataan vielä erikseen testitiimin toimesta. Testitiimin tehtävänä on luoda uudelle toiminnolle testiskenaariot ja varmistaa, että toiminto todella toimii, ennen kuin se dokumentoidaan ja julkaistaan.

5 Lopputulos

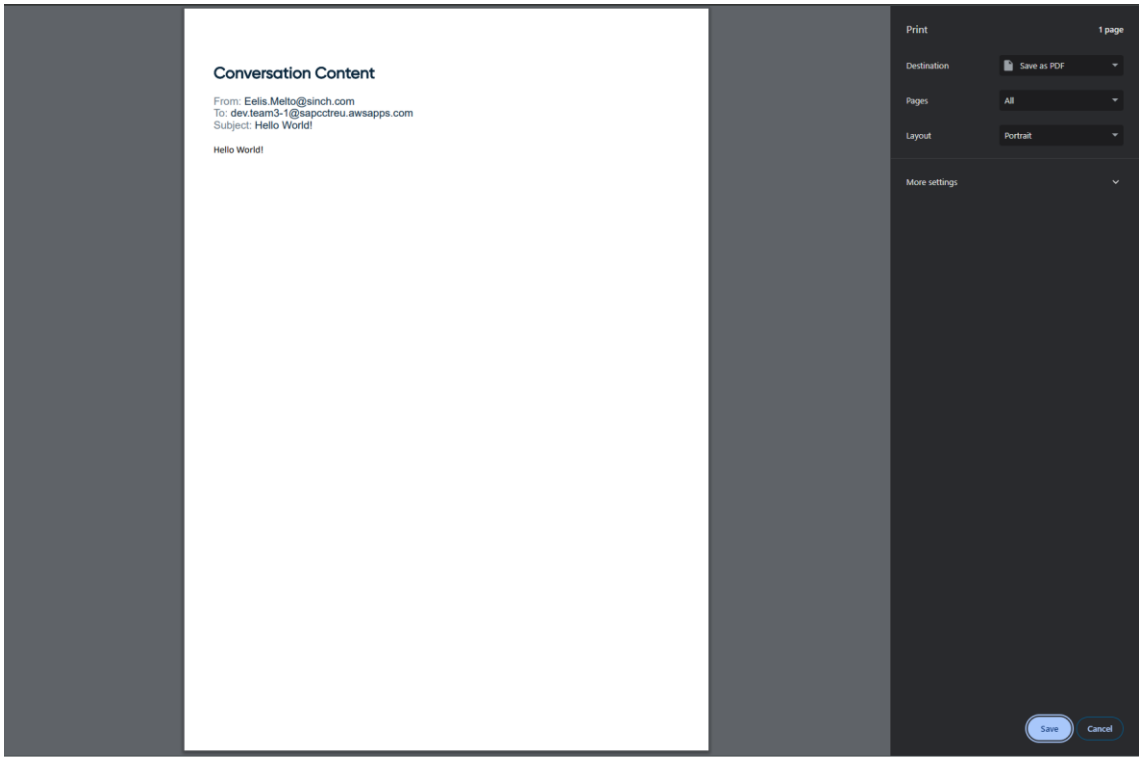
Lopputuloksena syntyi toiminto, joka täytti kehityspyynnön tarpeet. Sovelluksen käyttäjä pystyi nyt tulostamaan sähköpostien, tekstiviestien ja chattien sisällön PDF-muotoon uuden painikkeen avulla. Kuvassa 6 on esillä sovelluksen käyttöliittymä sähköpostikeskustelun käsittelyssä, jossa kehitetyn toiminnon painike näkyy oikeassa alakulmassa.



Kuva 6. Contact-Pro-yhteyskeskussovelluksen käyttöliittymä, jossa uusi PDF-tiedoston tulostava painike lisättynä oikeassa alakulmassa.

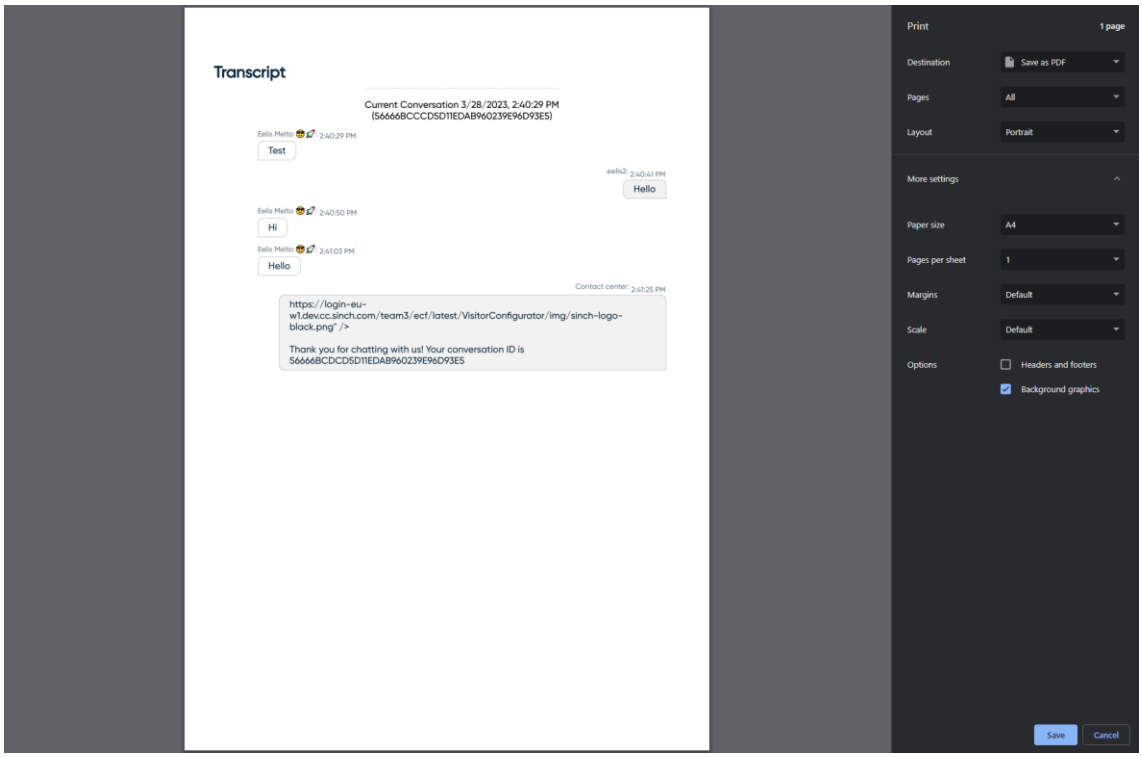
Kuvassa 7 on kuva tulostusikkunasta, kun sähköpostin sisältöä tulostetaan. Kuvan oikeassa alareunassa on painikkeet ”Save” ja ”Cancel”. Save-painikkeen avulla käyttäjä voi tallentaa PDF-muotoon tulostusikkunalla näkyvän sisällön.

Cancel-painikkeesta käyttäjä voi keskeyttää tulostuksen. Tulostusikkunan oikeassa yläreunassa näkyy tyylittelyyn liittyviä asetuksia.



Kuva 7. Sähköpostin tallentaminen PDF-muotoon tulostusikkunalla.

Kuvassa 8 tulostusikkunassa tulostetaan WhatsApp-keskusten sisältöä. Tulostusikkunan esittää WhatsApp-keskustelun samankaltaisena kuin se näkymällä esiintyi säilyttämällä esimerkiksi puhekuplien tyylit ja keskustelussa käytetyt emojit. Tässä tulostusikkunassa ”More settings” -asetukset on laajennettu, ja kaikki tulostusikkunan asetukset ovat esillä.



Kuva 8. WhatsApp-keskustelun tallentaminen PDF-muotoon tulostusikkunalla.

6 Bugit ja kehitysideat

Toiminnosta löytyi myöhemmin julkaisun jälkeen bugeja, jotka eivät ilmenneet kehityksessä tai testauksessa. Suurimmalla osasta käyttäjiä toiminto toimi kuitenkin hyvin ilman ongelmia. Yleisin ongelma oli, että keskustelun sisältö ei näkynyt tulostusikkunassa. Tämä johtui siitä, että alkuperäisessä versiossa tekstin fonttia ei esiladattu. Tämä myöhemmin korjattiin ja esimerkkikoodin 6 funktion link-elementin rel-ominaisuuteen lisättiin arvo ”preload”.

Toinen ongelma, mikä on ilmennyt uusien toimintojen ja muutosten tapahtuessa, on keskustelun tyylittelyn hajoaminen. Esimerkiksi chat-keskustelussa puhekupla ei ole oikealla, vaikka sen pitäisi tai keskustelun-ID ei ole keskellä vaan vasemmassa kulmassa. Koska ongelma ei kuitenkaan estä toimintoa toisin kuin sisällön katoaminen, luokitellaan se pieneksi ongelmaksi. Tyylittelyyn liittyvät ongelmat on kuitenkin korjattu toteutuksesta, vaikka niitä ei luokitella suureksi.

Kehitysidea liittyy insinöörityössä esitettyihin esimerkkikoodeihin 1 ja 2 kehyksen käyttämisestä. Koska kehyksen käyttäminen ei ollut vielä tuttua toiminnon kehityksen alussa, ei sen kaikkia ominaisuuksia käytetty. Esimerkiksi painikkeelle ei olisi tarvinnut luoda erillistä näkymää ja näkymälle omaa ohjainta vaan painikkeen ohjainta olisi voitu pelkästään muokata ja määrittää se keskustelunäkymän painikkeisiin suoraan. Tätä kutsutaan UI5:ssä ”Custom Control” -nimellä (14).

```
sap.ui.define([
    "sap/m/Button",
], function (Button) {
    "use strict";

    return Button.extend("com.myorg.myapp.control.PrintButton", {
        metadata: {
            events: {
            },
            properties: {
            },
            aggregations: {
            },
        },

        renderer: "sap.m.ButtonRenderer",

        onAfterRendering: function () {
            if (Button.prototype.onAfterRendering) {
                   Button.prototype.onAfterRendering.apply(this, argu-
                   ments);
            }

            this.attachPress({}, this.handlePress);
        },

        handlePress: function () {
              console.log(this.getText())
        },

        exit: function() {
            this.detachPress({}, this.handlePress);

            if (Button.prototype.exit) {
                Button.prototype.exit.apply(this, arguments);
            }
        }
    });
});
```

Esimerkkikoodi 14. Kehitysidea kuinka ”Custom Control” luotaisiin painikkeesta Open UI5 -kehyksellä. Koodissa on kuvattuna ohjain, joka perii painikkeen ohjaimen toiminnot ja määrittää niille uuden käyttäytymisen.

Esimerkkikoodissa 14 on esitettynä kehitysidean vaihtoehto kehyksen käytölle. Sen sijasta, että näkymälle luotaisiin oma ohjain, laajennetaan painikkeen alkuperäistä ohjainta. Tämän avulla erillistä näkymää ei tarvitsisi luoda, vaan keskustelunäkymään voitaisiin suoraan lisätä laajennettu painike (esimerkkikoodi 15). Esimerkkikoodin 15 XML-näkymän ominaisuus ”controllerName” on nyt vapaa käytettäväksi.

```
<mvc:View controllerName="com.myorg.myapp.controller.Main"
xmlns="sap.m" xmlns:mvc="sap.ui.core.mvc" xmlns:cc="
com.myorg.myapp.control">
      <cc:PrintButton text="Press Me!"/>
</mvc:View>
```

Esimerkkikoodi 15. Open UI5 XML -näkymä, jossa ”Custom Control” -painike määriteltynä.

Kehitysidea ei muuttaisi toimintoa millään tavalla käyttäjän näkökulmasta, mutta se vähentäisi kuitenkin käyttöliittymän viemää kuormitusta käyttäjän koneelta ja olisi tällöin täysin optimoitu toiminto UI5-kehykselle. Tarkoitus on muuttaa toiminto kehitysidean kaltaiseksi vuoden 2024 aikana, jos sen toteuttamiselle jää aikaa muiden uusien toimintojen ohella.

7 Yhteenveto

Insinöörityössä selvitettiin Sinch Finland Oy:n toimesta, kuinka rakennetaan Contact Pro -yhteyskeskussovellukseen toiminto, jonka avulla käyttäjä voi tulostaa tai tallentaa käyttöliittymän näkymän PDF-tiedostoksi ilman sovelluksen palvelimen hyödyntämistä. Työssä tehty toteutus on tehty UI5-kehystä käyttäen hyödyntämällä selaimen tulostusikkunan aukaisevaa print-metodia. Tulostusikkuna aukeaa uuden painikkeen painalluksella.

Painikkeen ohjaimelle on määritelty funktioita, joita kutsutaan vuoron perään, kun painiketta painetaan. Ensiksi kutsutaan funktiota, joka kopioi käyttöliittymän näkymän elementit. Seuraavaksi iframe-elementin alustavaa funktiota, joka asettaa kopioidut elementit iframe-elementin sisältöön ja mukauttaa iframe-elementin sovelluksen rakenteeseen. Lopuksi kutsutaan print-metodia, kun iframe-

elementin sisältö on latautunut. Tämän jälkeen käyttäjälle avautuu selaimen tulostusikkuna, jolla käyttäjä voi valita, tallennetaanko tulostusikkunan sisältö PDF-tiedostoksi vai tulostetaanko se suoraan tulostimella paperille.

Toiminnon funktioita testattiin QUnit-testeillä. QUnit-testeissä määriteltiin oletettu sisältö, joka palautuu funktioista. Jos funktio palauttaa jotain muuta kuin oletetun sisällön testi epäonnistuu. Tämän lisäksi toimintoa testasi yrityksen testitiimi. Vaikka toimintoa testattiin ja sen toiminallisuus oli hyvä julkaisua varten, löytyi siitä myöhemmin bugeja. Suurimalla osalla asiakkaista toiminto toimi ongelmitta, mutta muutamilla ilmeni toiminnon kanssa ongelmia. Esimerkiksi kopioitu sisältö ei näkynyt tulostusikkunassa eikä PDF-tiedostossa. Nämä ongelmat kuitenkin saatiin korjattua vuoden 2023 versiopäivitysten aikana.

Toteutuksesta syntyi kehitysidea, joka ei kohdistunut insinöörityössä esitettyihin funktioihin vaan käyttöliittymän kehyksen käyttämiseen. Kehitysidea ei muuttaisi käyttäjän näkökulmasta toiminnallisuutta, mutta se parantaisi yleisesti käyttöliittymän viemää kuormitusta käyttäjän koneesta ja olisi tällöin optimoitu kehykselle. Kehitysidea on tarkoitus toteuttaa vuoden 2024 aikana, mikäli sen tekemiseen jää aikaa.

Lähteet

1 **Mozilla.** Web API. *MDN Web Docs.* Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Web/API Luettu 31.10.2023.

2 **Mozilla.** Print-method. *MDN Web Docs.* Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Web/API/Window/print Luettu 31.10.2023.

3 **Mozilla.** Dokumentaatio. *PDF.js.* Verkkoaineisto. https://mozilla.github.io/pdf.js/ Luettu 31.10.2023.

4 **Hall, James.** Dokumentaatio. *jsPDF.* Verkkoaineisto. https://raw.githack.com/MrRio/jsPDF/master/docs/index.html Luettu 31.10.2023.

5 **Dillon, Andrew.** Dokumentaatio. *PDF-LIB.* Verkkoaineisto. https://pdf-lib.js.org/ Luettu 31.10.2023.

6 **Vieira, Rodrigo.** Dokumentaatio. *Print.js.* Verkkoaineisto. https://printjs.crabbly.com/ Luettu 31.10.2023.

7 **Adobe.** Create PDF from HTML. *Adobe document services.* Verkkoaineisto. https://developer.adobe.com/document-services/docs/overview/pdf-services-api/howtos/create-pdf/#create-a-pdf-from-static-html Luettu 31.10.2023.

8 **Mozilla.** Window API. *MDN Docs.* Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Web/API/Window Luettu 31.10.2023.

9 **Mozilla.** Printing. *MDN Web Docs.* Verkkoaineisto. https://developer.mozilla.org/en-US/docs/Web/Guide/Printing#print_an_external_page_without_opening_it Luettu 31.10.2023.

10 **WHATWG**. Print-method HTML-standard. Verkkoaineisto. https://html.spec.whatwg.org/multipage/timers-and-user-prompts.html#printing Luettu 05.11.2023.

11 **SAP.** MVC-arkkitehtuuri. *Open UI5.* 5. 11 2023 https://sdk.openui5.org/topic/91f233476f4d1014b6dd926db0e91070 Luettu 05.11.2023.

12 **SAP.** Painike ohjain. *Open UI5.* Verkkoaineisto.
https://sdk.openui5.org/api/sap.m.Button#overview
Luettu 05.11.2023.

13 **SAP.** XML-näkymä. *Open UI5.* Verkkoaineisto.
https://sdk.openui5.org/topic/91f292806f4d1014b6dd926db0e91070
Luettu 05.11.2023.

14 **SAP.** Custom Control. *Open UI5.* Verkkoaineisto.
https://sdk.openui5.org/topic/d12d2ee6a5454d799358d425f9e7c4db
Luettu 05.11.2023.