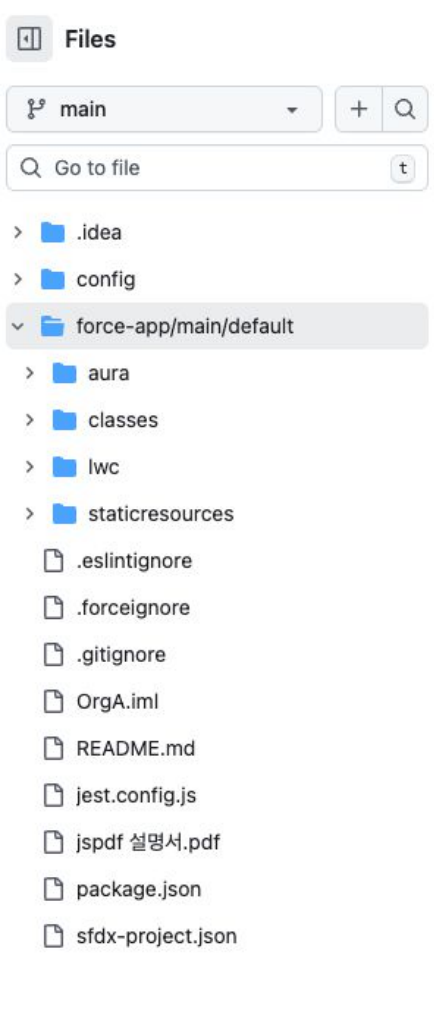
written by HJ.Yi

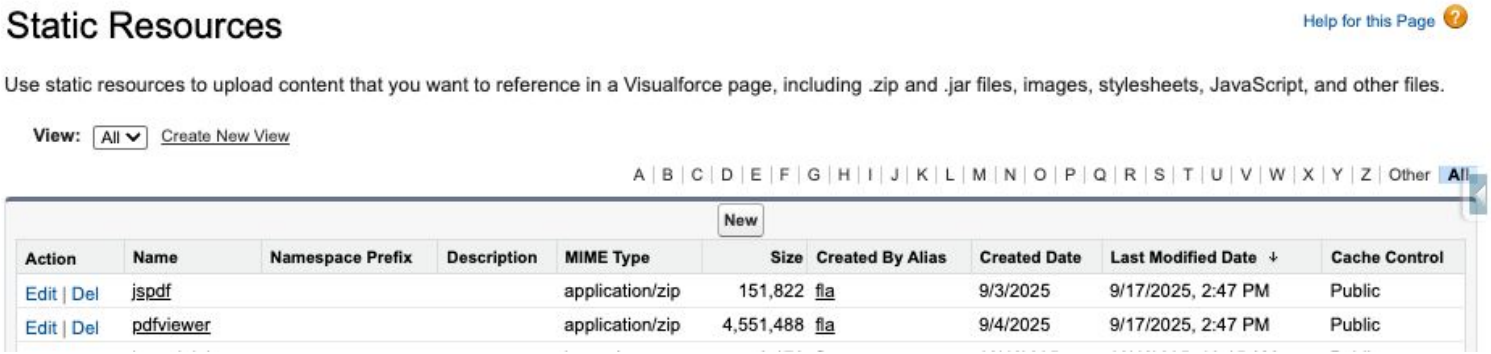
준비물

LWC	jsPdfCmp	소스 다운로드 : https://github.com/flashteker/jspdf (static js library는 zip으로 변경필요)
	commonShowToast	
Apex class	JsPdfController	
	JsPdfController_Test	
Static Resource	pdfviewer.zip	
	jspdf.zip	



Git에는 jsPdfCmp에 관련된 소스외에도 업데이트 된
설명서(“jspdf 설명서.pdf”)와
예제 그리고
예제에 사용된 폰트 들이 있다.

jsPdf js 라이브러리 등록



view의 type

TYPE	성격	설명
stack	container	content를 위에서 아래로 쌓는다.
horizontal	container	content를 왼쪽에서 오른쪽으로 쌓는다.
table	container	텍스트를 테이블 형식으로 담는다.
text	content	텍스트
image	content	이미지
line	content	라인
page	function	새로운 페이지를 추가한다.

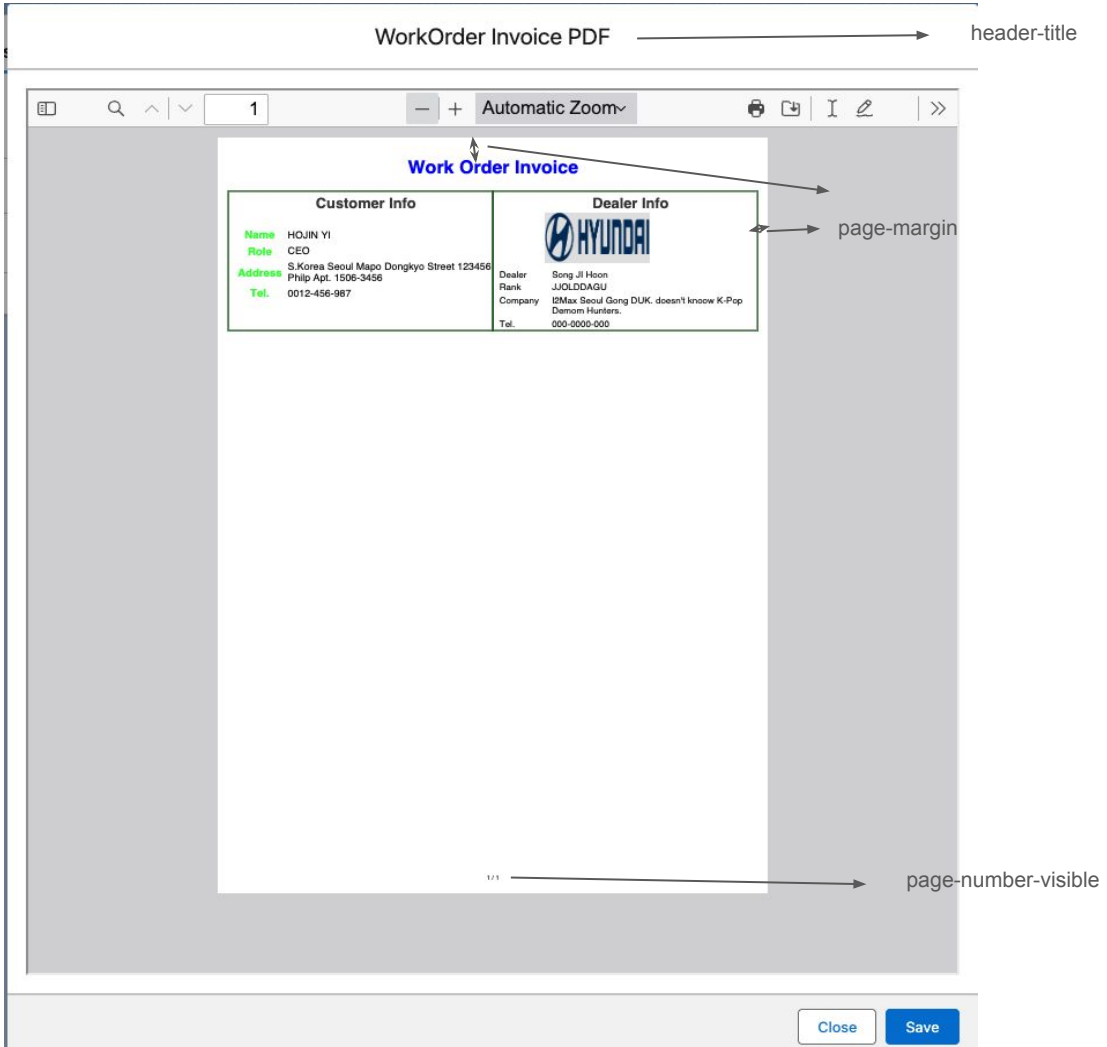
jsPdfCmp 사용 개요 - HTML

HTML

```
<template>
    <c-js-pdf-cmp
    header-title="WorkOrder Invoice PDF"
    ondrawready={handleOnDrawReady}
    record-id={recordId}
    file-name={pdfFileName}
    onclose={handleClose}
    onsave={handleSave}
    page-margin={pdfPageMargin}
    page-number-visible={pageNumberVisible}
    ></c-js-pdf-cmp>
</template>
```

* header-title : pdf 창의 타이틀을 나타낸다.
* page-margin : pdf전반의 여백을 지정한다.
* page-number-visible : 페이지 마다 하단에 페이지 번호를 나타낼지 여부를 지정한다.
* file-name : 저장시 파일 이름
* record-id : 저장할 오브젝트의 id
* onclose : 닫기 버튼이 눌러졌을 때 호출된다.
* onsave : 저장 버튼이 눌러졌을 때 호출된다.
* **ondrawready : 페이지에 그릴 준비가 완료되면 호출 된다.**

file-name과 record-id가 있을 경우 "저장" 버튼을 누르면 자동으로 저장한다.



jsPdfCmp 사용 개요 - js controller

js

```
import { LightningElement, api, track, wire } from 'lwc';
import { CloseActionScreenEvent } from 'lightning/actions';
export default class WoPdfCmp extends LightningElement {
    handleClose() {
        this.dispatchEvent(new CloseActionScreenEvent());
    }
    handleSave(event) {
        const kBase64PDF = event.detail.base64PDF;
        //do something for saving
        this.handleClose();
    }
    async handleOnDrawReady(event){
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        if(!kPdfCmp) return;
        const kBodies = [
            {type:"text", text:"Hello world"}
        ];
        kPdfCmp.startDraw({
            bodies:kBodies,
            footer:null,
            header:null
        })
    }

}
```

```
jsPdfCmp.startDraw({
  bodies:Array,
  footer:Object,
  header:Object
})
```

* 그리기 호출은 jsPdfCmd.startDraw(data)에서 시작한다.

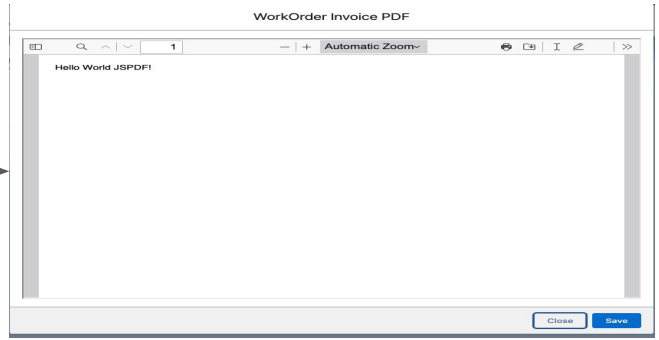
* data에는 bodies/footer/header 3개의 필드가 있다.(footer/header는 이후 설명)

* bodies는 배열 형식이며, view들이 들어갈 수 있다.

* 샘플 코드에서 알 수 있듯이 view는 특별한 객체를 사용하지 않고, 범용적인 오브젝트에 type으로 view를 결정한다.(**그래서 모든 view의 "type"필드 값은 필수**)

Sample - text

```
async handleOnDrawReady(event){
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
                {type:"text", text : "Hello World JSPDF!"}
            ];
        kPdfCmp.startDraw({
            bodies:kBodies
        })
    }
```

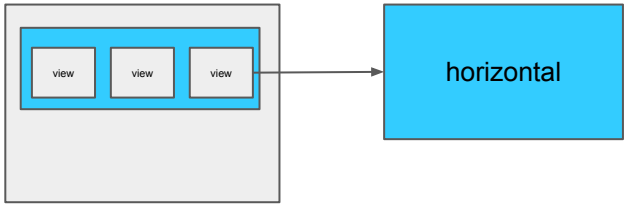


* 텍스트에는 텍스트에 관한 다양한 속성들이 가능한다.

```
async handleOnDrawReady(event){
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
                {type:'text',
                  text:'Hello World JSPDF!!',
                  styles:{
                     halign:"center",
                     fontSize:20,
                     fontStyle:"bold",
                     color:{r:0,g:0,b:250}
                  },
                  margin:{top:4}
                }
            ];
        kPdfCmp.startDraw({
            bodies:kBodies
        })
    }
```

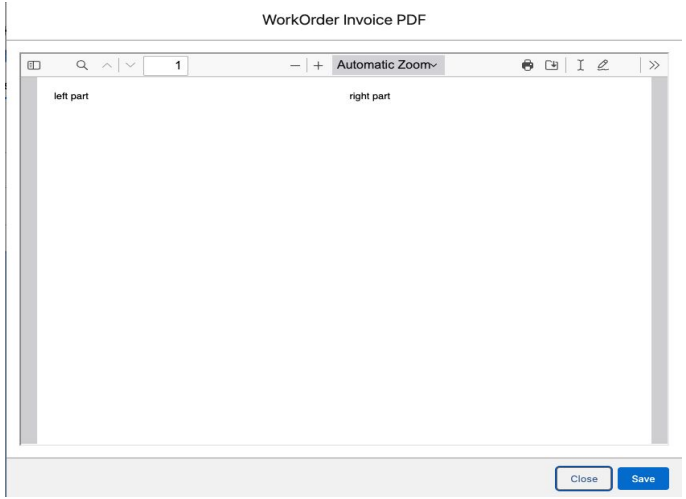
Sample (Layout) - horizontal

Horizontal은 좌에서 우로 뷰를 나열하는 레이아웃이다.
Layout(horizontal/stack)은 children이라는 배열을 가지며, 이 배열에는 뷰들이 나열된다.



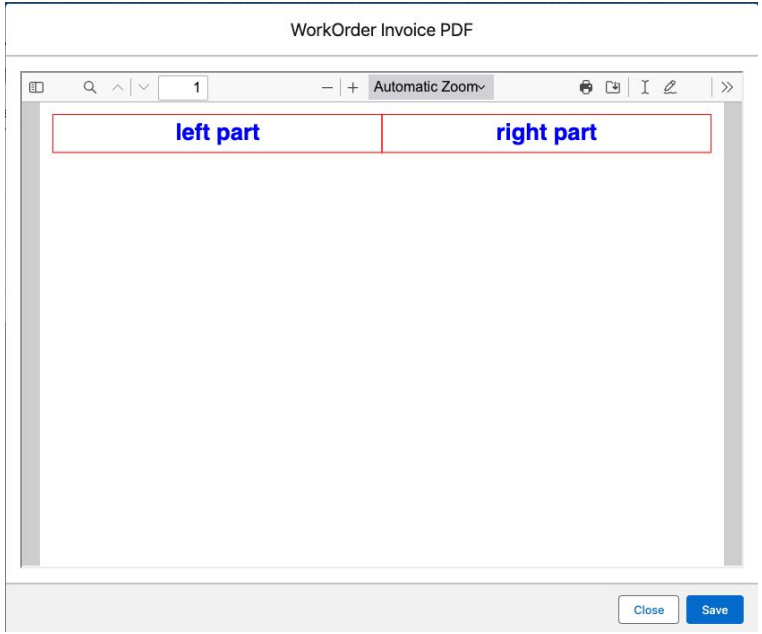

```
async handleOnDrawReady(event){
    const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
    const kBodies = [
        {
            type:'horizontal',

            children:[
                { type:"text", text:"left part"},
                {type:"text",text:"right part" }
            ]
        }
    ];
    kPdfCmp.startDraw({
        bodies:kBodies
    })
}
```

* horizontal layout에도 여러가지 속성을 지정할 수 있다.



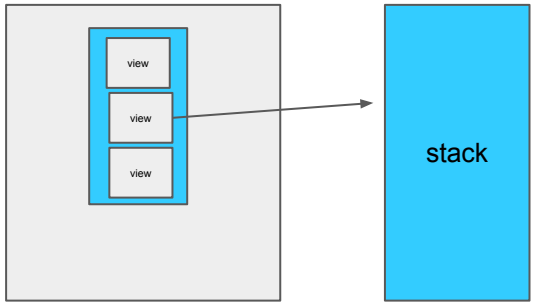
Sample - horizontal

```
async handleOnDrawReady(event){
    const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
    const kBodies = [
        {
            type:'horizontal',
            border:{thick:0.2,color:{r:255}},
            children:[
                {
                    type:"text",
                    text:"left part",
                    styles:{
                        halign:"center",
                        fontSize:20,
                        fontStyle:"bold",
                        color:{r:0,g:0,b:250}
                    }
                },
                {
                    type:"text",
                    text:"right part",
                    styles:{
                        halign:"center",
                        fontSize:20,
                        fontStyle:"bold",
                        color:{r:0,g:0,b:250}
                    }
                }
            ]
        }
    ];
    kPdfCmp.startDraw({
        bodies:kBodies
    })
}
```

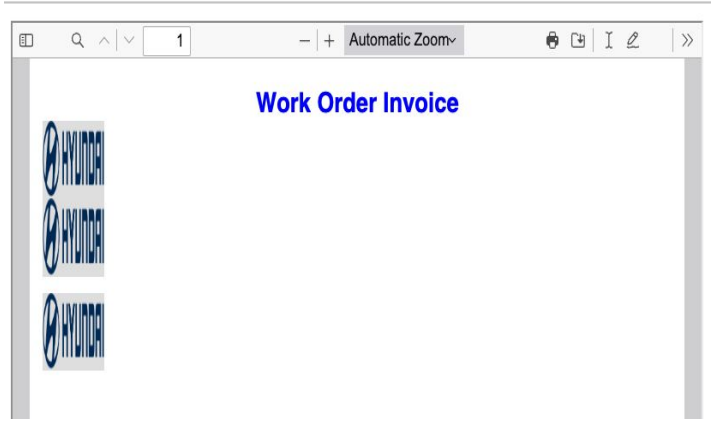


Sample(Stack) - stack

* stack은 위에서 아래로 차곡차곡 뷰들을 나열한다.




```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            {type:'stack',
                children:[
                    this.getHeaderTitle(),
                    this.getImageData(20,20),
                    this.getImageData(20,20),
                    this.getImageData(20,20, {top:2})
                ]
            }
        ];
        kPdfCmp.startDraw({bodies:kBodies})
    }
```



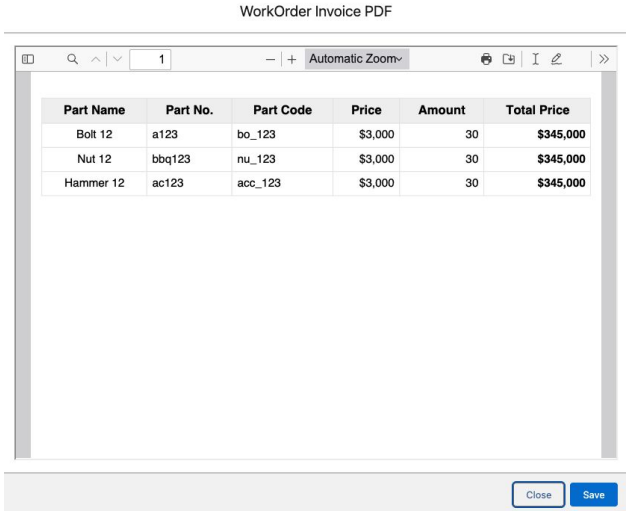
(!Tip) View data를 생성할 때 메소드를 이용하기를 권장한다. 이럴 경우 재사용이나 코드 관리가 편리하고, 뷰 구조 파악에 용이하다.

Sample - table

- table은 순수하게 텍스트를 그리드 형식으로 표시하기 위한 뷰이다.(이미지나 다른 뷰 불가)
- 각 셀마다 성질을 달리 줄 수 있게 하기 위하여 데이타들이 비교적 복잡하지만, head나 body를 제외하고는 모두 기본값으로 처리할 수 있다.

```
async handleOnDrawReady(event){
        const kPdfCmp =
this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getTableView()
        ];
        kPdfCmp.startDraw({
            bodies:kBodies
        })
    }
```

```
getTableView(){
        const kHeadBgColor = {r:240, g:240, b:240}
        const kHeadStyles = [
            {halign:"center", fontSize:13, bgColor:kHeadBgColor},
            {halign:"center", fontSize:13, bgColor:kHeadBgColor},
            {halign:"center", fontSize:13, bgColor:kHeadBgColor},
            {halign:"center", fontSize:13, bgColor:kHeadBgColor},
            {halign:"center", fontSize:13, bgColor:kHeadBgColor},
            {halign:"center", fontSize:13, bgColor:kHeadBgColor},
        ]
        const kBodyStyles = [
            {halign:"center", fontSize:12},
            {halign:"left", fontSize:12},
            {halign:"left", fontSize:12},
            {halign:"right", fontSize:12},
            {halign:"right", fontSize:12},
            {halign:"right", fontSize:12, fontStyle:"bold"},
        ];
        return{
            type:'table',
            border:{thick:0.2, color:{r:230, g:230, b:230}},
            margin:{top:6},//space from parent
            cellPadding:2,//셀과 텍스트간의 간격 head, body 구분하지 않음
            headStyles:kHeadStyles,
            bodyStyles:kBodyStyles,
            head:['Part Name','Part No.', 'Part Code', 'Price', 'Amount', 'Total Price'],
            body:[
                ['Bolt 12', 'a123', 'bo_123', '$3,000', '30','$345,000'],
                ['Nut 12', 'bbq123', 'nu_123', '$3,000', '30','$345,000'],
                ['Hammer 12', 'ac123', 'acc_123', '$3,000', '30','$345,000']
            ]
        }
    }
```



Sample - 헤더가 없는 table

- head를 지정하지 않으면 헤더 없이 그려진다.(이 원리를 이해하면 다양한 형식의 표를 만들 수 있다.)

```
async handleOnDrawReady(event){
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getTableView(),
            this.getSummaryTableData()
        ];
        kPdfCmp.startDraw({
            bodies:kBodies
        })
    }

.....
getSummaryTableData(){
        const kBodyStyles = [
            {fontSize:12, color:{b:255}},
            {halign:"right", fontSize:12, color:{r:255}}
        ];
        return{
            type:'table',
            margin:{left:100},
            border:{thick:0.2, color:{r:200, g:200, b:0}},
            bodyStyles:kBodyStyles,
            body:[
                ['Total Price', "$1,000,000"],
                ['Total TAX', "$4,000"],
                ['Total Amount', "$1,040,000"]
            ]
        }
    }
```

Part Name	Part No.	Part Code	Price	Amount	Total Price
Bolt 12	a123	bo_123	$3,000	30	$345,000
Nut 12	bbq123	nu_123	$3,000	30	$345,000
Hammer 12	ac123	acc_123	$3,000	30	$345,000

Total Price	$1,000,000
Total TAX	$4,000
Total Amount	$1,040,000

Sample - image

```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getImageData(10,20,{left:20, top:0}),
            this.getImageData(20,40,{left:10, top:5})
        ];
        kPdfCmp.startDraw({
            bodies:kBodies
        })
    }

async preloadImage(imgUrl) {
        try {
            const response = await fetch(imgUrl);
            const blob = await response.blob();
            const base64 = await new Promise((resolve, reject) => {
                const reader = new FileReader();
                reader.onloadend = () => resolve(reader.result);
                reader.onerror = reject;
                reader.readAsDataURL(blob);
            });
            return  base64 ;
        } catch (e) {
            console.error('Failed to load damage legend image:', imgUrl, e);
            return null;
        }
    }

getImageData(width, height, margin){
                return {
            type:"image",
            margin:margin,
            image:{src:this.logoBase64, w:width, h:height}
        }
    }
```

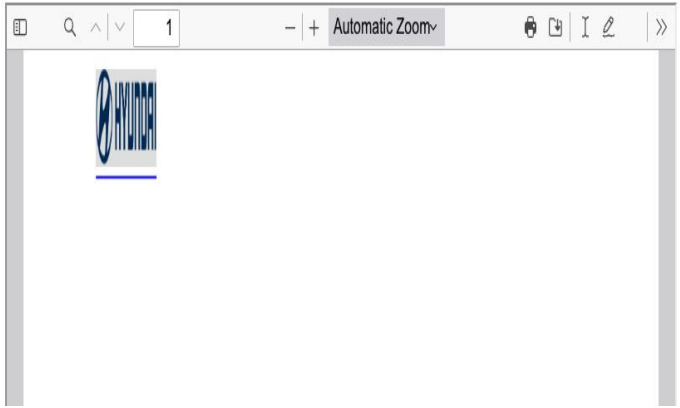
* image는 base64로 인코딩된 데이타만 사용할 수 있다.

*** png 보다는 jpg 사용을 권장한다.(LWC에서 변환할때 시간 차이가 큼)**

Sample - line

```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getImageData(20,20,{left:10, top:0}),
            {
                type:'line',
                border:{thick:0.5, color:{r:0, g:0, b:255}},
                margin:{left:20, top:2},
                width:20
             },
        ];
        kPdfCmp.startDraw({bodies:kBodies})
    }
```

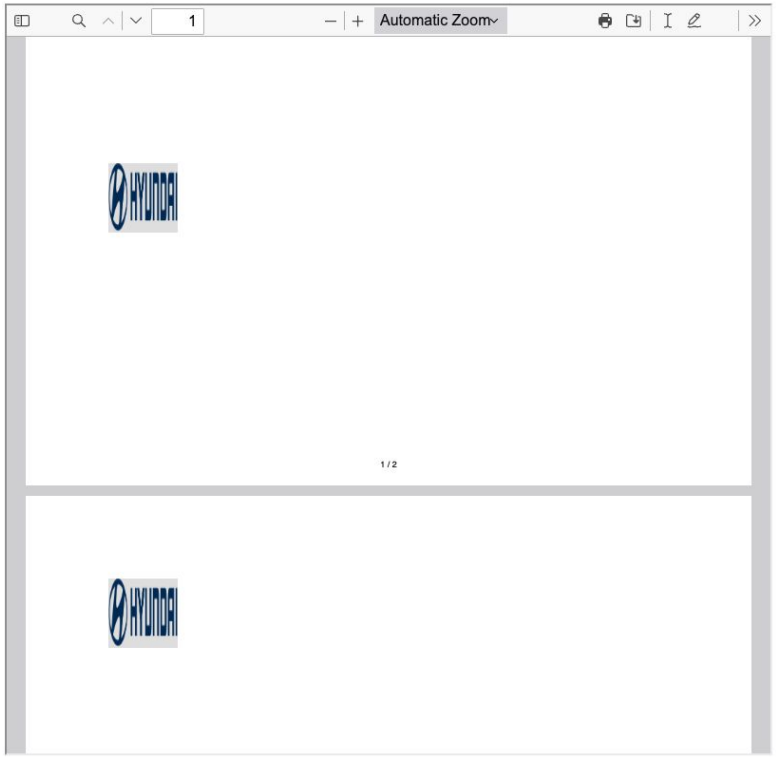
* image에서 2 픽셀 아래에 파란색 라인을 그린다



Sample - page

```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getImageData(20,20,{left:10, top:100}),
            {type:'page'},
            this.getImageData(20,20,{left:10, top:10}),
        ];
        kPdfCmp.startDraw({bodies:kBodies})
    }
```

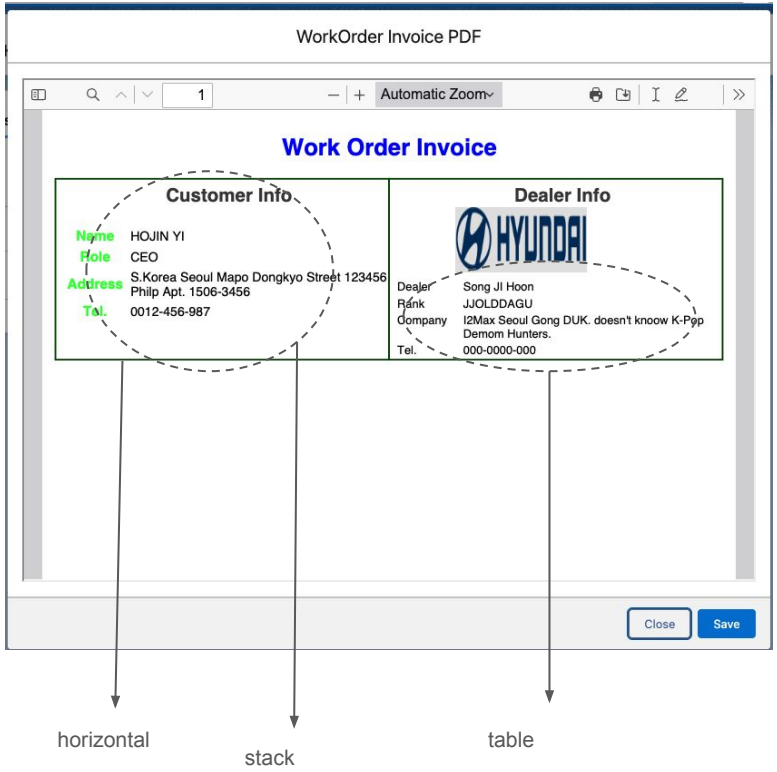
* 새로운 페이지를 추가한다.
*** 최상위 레이아웃에서만 사용할 수 있으며, 내부 레이아웃에서는 사용할 수 없다.(사용하더라도 무시된다.)**



Container/Component 모델

* Layout은 또 다른 Layout을 자식뷰로 가질 수 있으며, 이를 이용하여 복잡한 뷰를 생성할 수 있다.

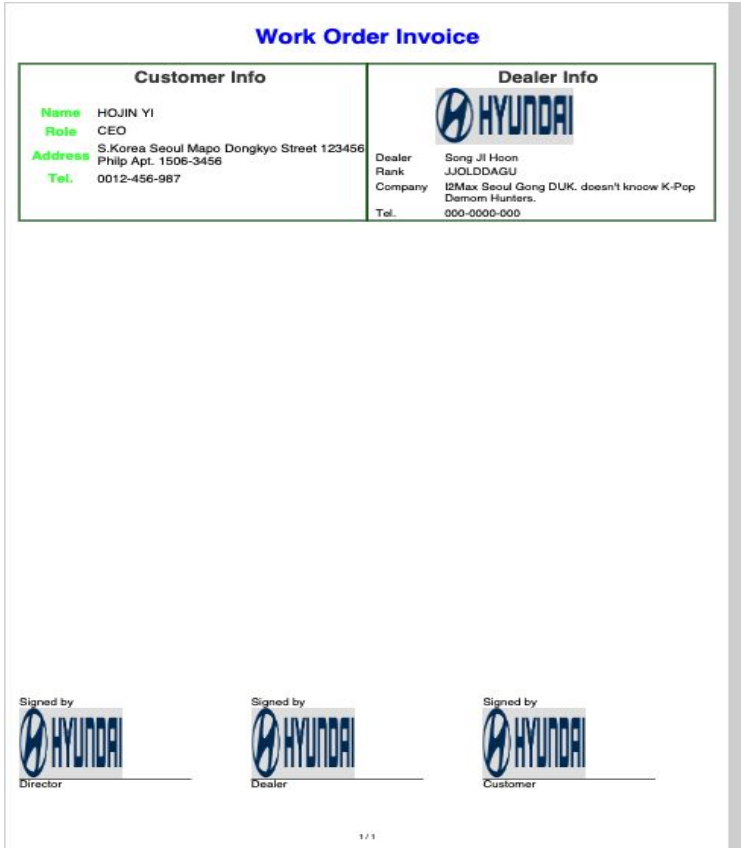
```
async handleOnDrawReady(event){
      this.logoBase64 = await this.preloadImage(hyundai_logo);
      const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
      const kBodies = [
         this.getHeaderTitle(),
         this.getHeaderView()
      ];
      kPdfCmp.startDraw({bodies:kBodies})
   }
…
getHeaderView(){
      return {
        type: 'horizontal',
        margin:{top:5},
        border:{thick:0.5, color:{r:0, g:70, b:0}},
        children:[
           {
              type:'stack',
              children:[
                 this.getTextData('Customer Info', {left:2, top:2},
{halign:"center",fontSize:16,fontStyle:"bold",color:{r:60,g:60,b:60}}),
                 this.getCustomerInfoTableView()
              ]
           },

           {
              type : 'stack',
              children :[
                 this.getTextData('Dealer Info', {left:2, top:2},
{halign:"center",fontSize:16,fontStyle:"bold",color:{r:60,g:60,b:60}}),
                 this.getImageData(40,20,{left:10}),
                 this.getDealerInfoTableView()
              ]
           }
        ]
     }
   }
```



footer

* footer는 제일 마지막 페이지 하단에 표시되는 영역을 나타내는 뷰이다.
* height 및 child 필들를 가지며, child는 view이다.
* 페이지 마다 반복되지 않는다.

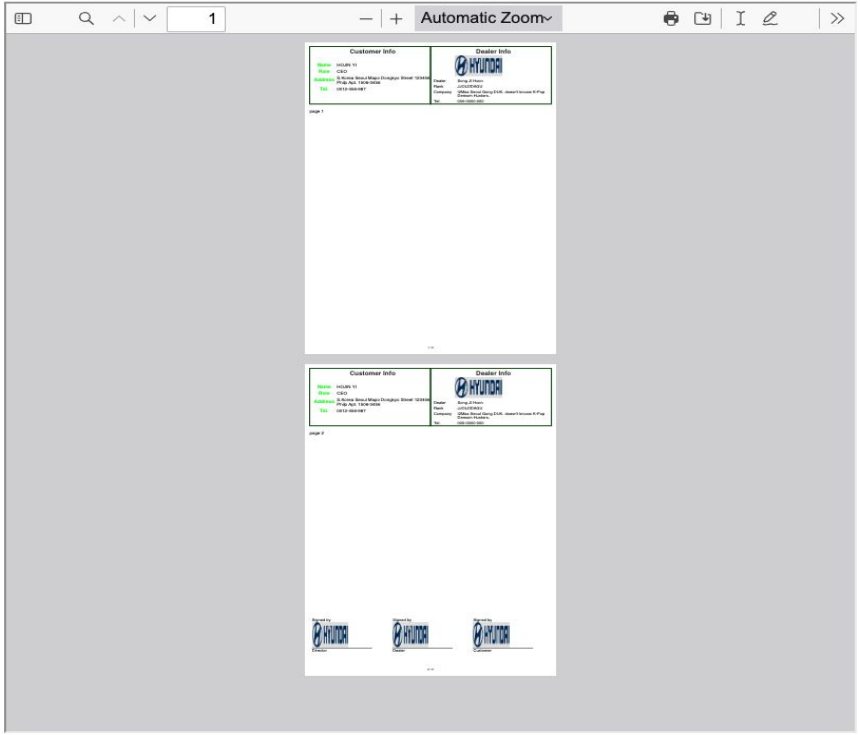
```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getHeaderTitle(),
            this.getHeaderView()
        ];
        kPdfCmp.startDraw({
            bodies:kBodies,
            footer:{height:45, child:this.getFooterView()}
        })
    }
...
getFooterView(){
        return {
            type:'horizontal',
            children:[
                this.getSignatureView('Director'),
                this.getSignatureView('Dealer'),
                this.getSignatureView('Customer')
            ]

        }
    }
getSignatureView(title){
        return {
            type:'stack',
            margin:{left:0},
            children : [
                {type:'text', styles:{fontSize:10}, text:"Signed by "},
                this.getImageData(30,25),
                {type:'line', border:{thick:0.5}, margin:{top:0},width:50},
                {type:'text', styles:{fontSize:10}, text:title}
            ]
        }
    }
```



header

* header는 페이지마다 맨 상단에 표시되는 뷰이다.
* footer와 마찬가지로 height와 child속성을 가진다.
* 왜 옵션없이 페이지 마다 반복 되냐규? 반복되지 않는 헤더라면 그냥 **bodies**의 맨 처음 뷰 위치에 놓으면 된다.

```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getTextView("page 1"),
            {type:"page"},
            this.getTextView("page 2")
        ];
        kPdfCmp.startDraw({
            bodies:kBodies,
            footer:{height:45, child:this.getFooterView()},
            header:{height:60, child:this.getHeaderView()}
        })
    }
```



TIP - 매우 긴 장문을 표시할 때

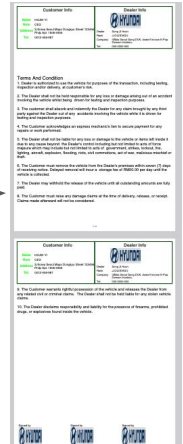
* jsPdf의 가장 큰 단점 2개
1. 화면에 그린 이후 수정 불가
2. header나 footer 영역을 자동으로 제외하고 그리기 불가능

이러다 보니, header가 있을 경우 매우 긴 장문(예를 들어 라이센스 문구)을 기입할 때 많은 여백을 남기고 다음 페이지로 넘어간

그래서 긴문장은 배열로 잘라서 text view로 처리하기를 권장한다 .

```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getTextView("Terms And Condition",{top:30}, {fontSize:18}),
            this.getTextView(this.txtTermsList.join('\n\n'),{top:20},{fontSize:14})

        ];
        kPdfCmp.startDraw({
            bodies:kBodies,
            footer:{height:45, child:this.getFooterView()},
            header:{height:60, child:this.getHeaderView()}
        })
    }
```

```
async handleOnDrawReady(event){
        this.logoBase64 = await this.preloadImage(hyundai_logo);
        const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
        const kBodies = [
            this.getTextView("Terms And Condition",{top:30}, {fontSize:18})
        ];
        this.txtTermsList.forEach(txt => {
            const kTxtData = this.getTextView(txt+'\n',{top:0},{fontSize:14});
            kBodies.push(kTxtData);
        })
        kPdfCmp.startDraw({
            bodies:kBodies,
            footer:{height:45, child:this.getFooterView()},
            header:{height:60, child:this.getHeaderView()}
        })
    }
```

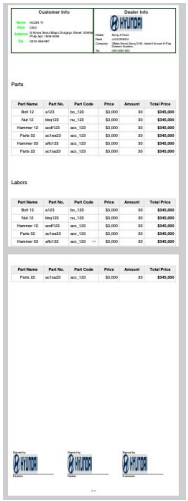


TIP - table view의 경우 레코드(행)가 많아질때

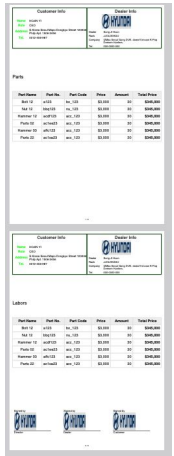
* table은 기본적으로 자동으로 페이지를 나누어준다 하지만 header나 footer가 있을 경우 레코드가 길어 페이지가 나누어질 때 header나 footer(페이지 표시 영역포함) 영역을 침범한다.

이럴 경우, 명시적으로 페이지를 나누어 (type:"page") 이를 피한다.

```
async handleOnDrawReady(event){
      this.logoBase64 = await this.preloadImage(hyundai_logo);
      const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
      const kBodies = [
          this.getTextView("Parts",{top:30}, {fontSize:18}),
          this.getTableView(),
          this.getTextView("Labors",{top:30}, {fontSize:18}),
          this.getTableView()
      ];

      kPdfCmp.startDraw({
          bodies:kBodies,
          footer:{height:45, child:this.getFooterView()},
          header:{height:60, child:this.getHeaderView()}
      })
  }
```



```
async handleOnDrawReady(event){
      this.logoBase64 = await this.preloadImage(hyundai_logo);
      const kPdfCmp = this.template.querySelector('c-js-pdf-cmp');
      const kBodies = [
          this.getTextView("Parts",{top:30}, {fontSize:18}),
          this.getTableView(),
          {type:"page"},
          this.getTextView("Labors",{top:30}, {fontSize:18}),
          this.getTableView()
      ];

      kPdfCmp.startDraw({
          bodies:kBodies,
          footer:{height:45, child:this.getFooterView()},
          header:{height:60, child:this.getHeaderView()}
      })
  }
```



범용 속성

Property	Data Type	설명
styles	Object	텍스트 및 font 속성 (halign, valign, fontSize, fontStyle, color)
bodyStyles / headStyles	Object	table view 의 cell의 텍스트 및 font 속성 (halign, valign, fontSize, fontStyle, color, cellWidth)
margin	Object	{left:, top:, right:, bottom:}. 상위 컨테이너 내에서의 자신 위치
border	Object	{thick:, color:}
color	Object	{r, g, b}
fontSize	Number	font size
text	String	“text’ view에서 text 내용
left/ top/right/ bottom	Number	상위 뷰에서의 나의 상대적인 위치
halign	String	“left”(default) / “center” /”right”
valign	String	“top” / “middle”(default)/”bottom”
fontStyle	String	“bold”, “normal”(default)
r/g/b	Number	0 ~ 255
width/height	Number	너비 및 높이
cellWidth	Number	table에서 한 컬럼의 너비를 지정한다.
cellPadding	Number	table cell내의 패딩
body	array	1차 배열 [str, str ….] . 테이블 헤더의 필드 값들을 나열. table에서만 사용
head	array	2차 배열 [[str, str…], [str, str…],....]. 각 행들의 집합.. table에서만 사용
image	Object	src, width, height
src	String	image base64 string data

각 View의 속성들 - stack

Property	Data Type	설명
type	**String 상수**	**‘stack’**
margin	Object	
border	Object	
children	Array	child Views

```
{
        type:'stack',
        margin:{left:2, top:10, right:2, bottom:2},
        border:{thick:0, color:{r:255}},
        children : [
            {type:'text', styles:{fontSize:10}, text:"Signed by "},
            this.getImageData(30,25),
            {type:'line', border:{thick:0.5}, margin:{top:0},width:50},
            {type:'text', styles:{fontSize:10}, text:title}
        ]
    }
```

각 View의 속성들 - horizontal

Property	Data Type	설명
type	**String 상수**	**‘horizontal’**
margin	Object	
border	Object	
children	Array	child Views

```
{
        type:'horizontal',
        margin:{left:2, top:3},
        border:{thick:0.2,color:{r:255}},
        children:[
            this.getSignatureView('Director'),
            this.getSignatureView('Dealer'),
            this.getSignatureView('Customer')
        ]

    }
```

각 View의 속성들 - page

Property	Data Type	설명
type	**String 상수**	**‘page’**

```
{
        type:'page',
}
```

각 View의 속성들 - line

Property	Data Type	설명
type	**String 상수**	**‘line’**
margin	Object	
border	Object	
width	Number	
height	Number	

```
{
        type:'line',
        border:{thick:0.5},
        margin:{top:0},
        width:50
    }
```

각 View의 속성들 - image

Property	Data Type	설명
type	**String 상수**	**‘image’**
margin	Object	
border	Object	
image	Object	

```
{
        type:"image",
        margin:margin,
        image:{src:this.logoBase64, width:kWidth, height:kHeight}
    }
```

각 View의 속성들 - text

Property	Data Type	설명
type	**String 상수**	**‘text’**
margin	Object	
border	Object	
styles	Object	
text	String	

```
{
 type:'text',
   styles:{fontSize:10},
     text:"Signed by "
}
```

각 View의 속성들 - table

Property	Data Type	설명
type	**String 상수**	**'table'**
margin	Object	
border	Object	
bodyStyles	Object	
headStyles	Object	
cellPadding	Number	
head	Array	1차 배열 [str, str ….] . 테이블 헤더의 필드 값들을 나열
body	Array	2차 배열 [[str, str…], [str, str…],….]. 각 행들의 집합.

```
const kHeadBgColor = {r:240, g:240, b:240}
    const kHeadStyles = [
        {halign:"center", fontSize:14, bgColor:kHeadBgColor},
        {halign:"center", fontSize:14, bgColor:kHeadBgColor},
        {halign:"center", fontSize:14, bgColor:kHeadBgColor},
        {halign:"center", fontSize:14, bgColor:kHeadBgColor},
        {halign:"center", fontSize:14, bgColor:kHeadBgColor},
        {halign:"center", fontSize:14, bgColor:kHeadBgColor},
    ]
    const kBodyStyles = [
        {halign:"center", fontSize:14},
        {halign:"left", fontSize:14},
        {halign:"left", fontSize:14},
        {halign:"right", fontSize:14},
        {halign:"right", fontSize:14},
        {halign:"right", fontSize:14, fontStyle:"bold"},
    ];
```

```
{
    type:'table',
    border:{thick:0.2, color:{r:230, g:230, b:230}},
    margin:{top:16},//space from parent
    cellPadding:2,//셀과 텍스트간의 간격 head, body 구분하지 않음
    headStyles:kHeadStyles,
    bodyStyles:kBodyStyles,
    head:['Part Name','Part No.', 'Part Code', 'Price', 'Amount', 'Total Price'],
    body:[
        ['Bolt 12', 'a123', 'bo_123', '$3,000', '30','$345,000'],
        ['Nut 12', 'bbq123', 'nu_123', '$3,000', '30','$345,000']
    ]
}
```

버튼 이벤트 처리 - 창 닫기 및 저장하기

jsPdfCmp에 등록한 이벤트 핸들러를 이용한다.

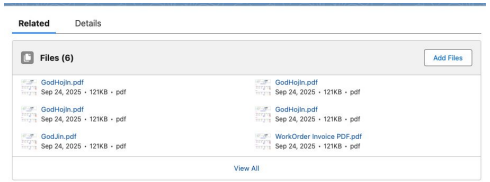
```
<template>
    <c-js-pdf-cmp
    header-title="WorkOrder Invoice PDF"
    ondrawready={handleOnDrawReady}
    record-id={recordId}
    file-name={pdfFileName}
    onclose={handleClose}
    onsave={handleSave}
    page-margin={pdfPageMargin}
    page-number-visible={pageNumberVisible}
    ></c-js-pdf-cmp>
</template>
```

```
  import { CloseActionScreenEvent } from 'lightning/actions';
….
handleClose() {
        this.dispatchEvent(new CloseActionScreenEvent());
    }

    handleSave(event) {
        const kBase64PDF = event.detail.base64PDF;
        //do something for saving
        this.handleClose();
    }
```

* 창 닫기는 CloseActionScreenEvent를 이용한다.

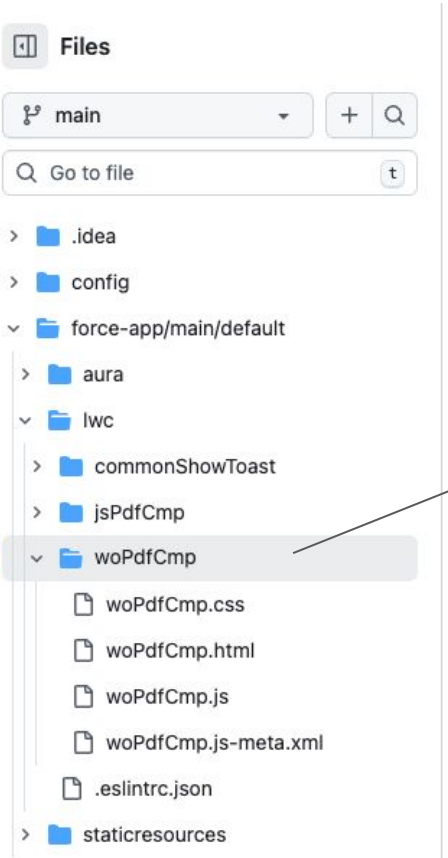
* 저장은 event.detail.base64PDF를 이용한다. 이 변수에는 pdf를 base64 string 형식으로 담고 있다.

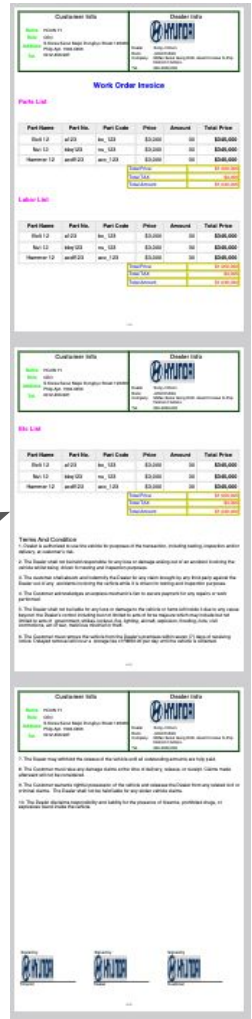
*** 만약 “record-id”와 “file-name”이 지정되면 자동으로 파일을 저장한다.**



샘플 LWC 콤포넌트

https://github.com/flashteker/jspdf 에 접속하면 jsPdfCmp를 이용한 woPdfCmp 소스가 있다.
다양한 View뿐만 아니라, 창 닫기 및 저장하기에 관한 샘플 소스들이 있다.

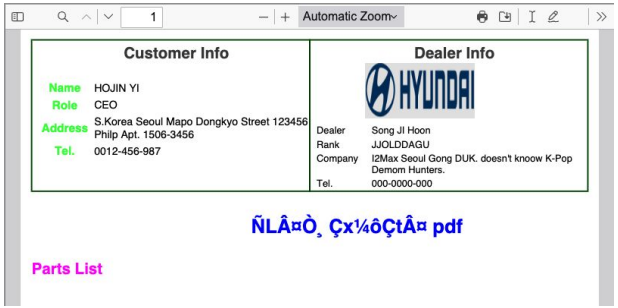




한국어 적용 및 커스텀 폰트 처리

한국어 적용 및 커스텀 폰트 처리 - 0. 개요

- jsPdf라이브러리는 서버나 브라우저의 시스템 폰트를 사용하지 못하고, 내부에 코드로 등록된 폰트만 사용할 수 있다.
- 등록할 font는 Base64 타입으로 인코딩 되어야 한다.(이런…)
- 현재 내장되어 있는 폰트는 “Helvetica”(default), “Times”, “Symbol”, “Courier” 이며 이들은 모두 영어이외의 문자를 지원하지 않는다. 그래서 만약 한글을 기입하면 문자가 깨져서 보인다.




```
getHeaderTitle(){
      return {
        type:'text',
        text:'테스트 인보이스 pdf',
        styles:{
            halign:"center",
            fontSize:20,
            fontStyle:"bold",
            color:{r:0,g:0,b:250}
        },
        margin:{top:4}
      }
  }
```

- 하지만 jsPdf에 다른 폰트를 적용하는 방법은 만만치 않다. 간략하게 프로세서를 정리하자면

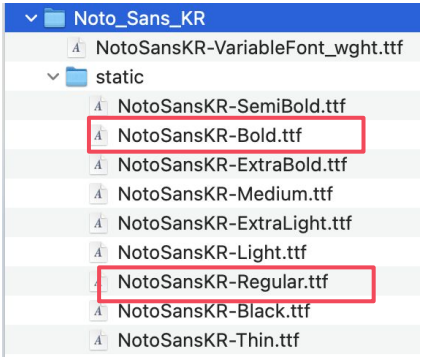
1) 폰트를 base64로 변경
2) 이 데이타를 폰트에 등록하는 코드를 추가하여 js파일로 저장
3) static resource에 등록
4) 로딩 및 실행하여 등록

- 폰트는 ttf 타입의 폰트만 지원하며, 만약 스타일이 normal과 더불어 bold체도 필요하다면 2개의 폰트 모두를 등록해야 한다.
(현재로서는 italic 스타일은 지원하지 않는다. 수요를 보아서 필요할 경우 추가 작업 필요)

- jsPdfCmp는 또 다른 폰트를 사용할 수 있는 환경을 제공한다. 원하는 폰트를 static resource에 등록하고 사용할 수 있는 방법을 다음 장부터 설명한다.
(jsPdfCmp는 하나의 폰트만을 지원한다. 그러기에 custom font는 하나만 등록할 수 있다. 만약 2개 이상의 폰트를 지원하고 싶다면 jsPdfCmp 코드를 직접 수정해야 한다.)

한국어 적용 및 커스텀 폰트 처리 - 1. 폰트파일 준비

- 폰트 파일은 ttf 파일로 준비한다. 만약 pdf에서 bold체도 사용한다면 bold체 파일도 준비한다.
(한글을 지원하는 대표적인 무료폰트는 구글에서 제공하는 “NotoSansKR”이 있으며 그외에도 검색하면 많이 나온다. 우리는 NotoSansKR을 사용할 것이다.)



- 우리가 사용할 폰트 파일은 NotoSansKR_Regular.ttf와 NotoSansKR_Bold.ttf 2개의 파일을 등록할 것이다.

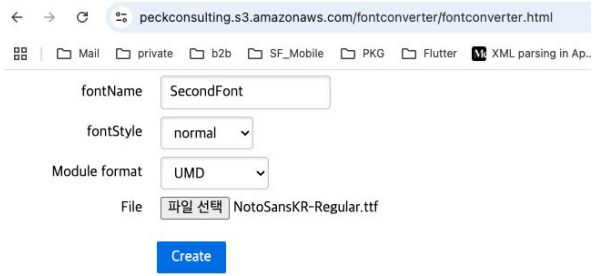
한국어 적용 및 커스텀 폰트 처리 - 2. 폰트를 js 파일로 변환

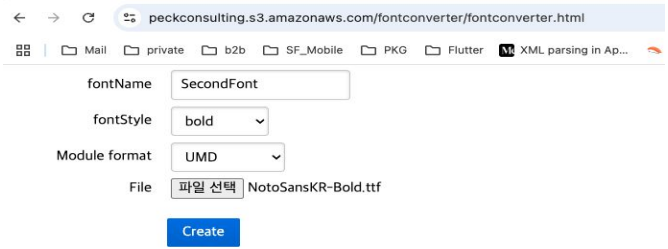
- 약간 까다롭기는 하지만 다음의 순서를 따른다.

1. 변환 사이트 오픈(https://peckconsulting.s3.amazonaws.com/fontconverter/fontconverter.html)

2. “파일선택"에서 변환을 원하는 파일을 선택한다.

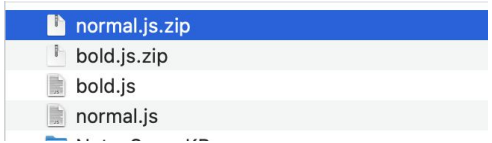
3. fontName과 fontStyle을 지정한다.





- 그림에서 보는 바와 같이 Module format은 “UMD”로 선택
- fontName은 normal이나 bold나 동일(fontName은 꼭 기억하고 있어야 한다.)
(위의 그림에서 우리가 사용하는 폰트는 “SecondFont”이고 style은 “normal”과 “bold” 로 나누어진다.)

4. Create하면 js파일이 만들어 지는데, 만들어진 후 각각의 파일 이름은 “normal.js”, “bold.js”로 변경한다.
5. Static Resource에 등록하기 위하여 2개의 파일을 zip 파일로 압축한다.
(Js파일은 폰트를 base64로 변환 했기에 파일 사이즈가 대부분 크다. static resource는 5Mb 이상을 허용하지 않기 때문에 zip으로 압축하는 것으로 통일.)



한국어 적용 및 커스텀 폰트 처리 - 3. Static Resources에 등록

1. 변환된 zip파일을 다음 그림과 같이 등록한다.
- “pdf_font_normal”과 “pdf_font_bold”

New

Action	Name	Namespace Prefix	Description	MIME Type	Size	Created By Alias	Created Date	Last Modified Date ↓	Cache Control
Edit | Del	pdf_font_normal		pdf에 사용되는 폰트 normal style (주로 영어이외의 문자가 사용될 때 사용한다.)	application/zip	3,755,591	fla	9/24/2025	9/24/2025, 6:15 PM	Public
Edit | Del	pdf_font_bold		pdf에 사용되는 폰트 bold체 (주로 영어이외의 문자가 사용될 때 사용한다.)	application/zip	3,780,906	fla	9/24/2025	9/24/2025, 6:15 PM	Public

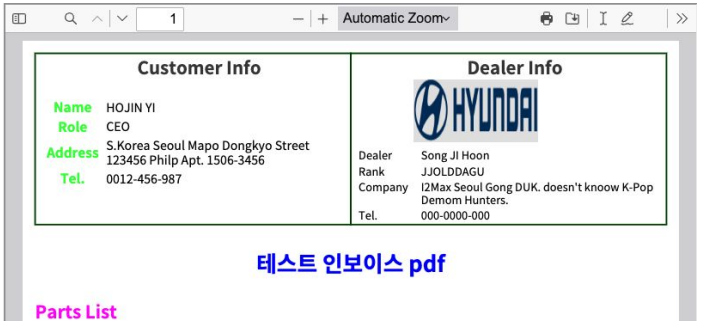
- 그림에서 보듯이 우리가 등록할 수 있는 여분의 폰트는 **1개이다.(파일은 normal/bold 2개)**

한국어 적용 및 커스텀 폰트 처리 - 4. 적용

```
<template>
    <c-js-pdf-cmp
    header-title="WorkOrder Invoice PDF"
    ondrawready={handleOnDrawReady}
    record-id={recordId}
    file-name={pdfFileName}
    onclose={handleClose}
    onsave={handleSave}
    page-margin={pdfPageMargin}
    page-number-visible={pageNumberVisible}
    font-name="SecondFont"
    ></c-js-pdf-cmp>
</template>
```

- 콤포넌트 세팅시 “font-name”을 지정한다.
- 이 font-name은 “2. 폰트를 js 파일로 변환” 단계에서 지정했던 “fontName”이다.

- 이제 실행해서 보면 한글이 깨지지 않고 잘 나온다.
(custom font를 지정하면 로딩하는 시간이 좀더 걸린다.)



한국어 적용 및 커스텀 폰트 처리 - 5. 마무리하며

1. 커스텀 폰트를 지정하는 것이 까다로워 최소한의 자동화로 편의성을 높이려고 하였고, 이로 인해 제약 사항들이 많다.

2. **jsPdfCmp는 단일 폰트 적용 구조이다.** 2개 이상의 폰트 설정 불가
이후 2개 이상의 폰트 지정이 필요하다는 수요가 있을 경우 추가의 작업으로 해결 해야 한다.

3. 한국어 외에도 인도네시아어를 적용하려면 커스텀 폰트를 지정해야 할 것이다.