

```

1 // Rueoiarhea[r[,eau[rmeagrte8n[mapgvvy[akt,lamtaeubmpeuay[rarg[earue[gat[g[]]]
2 import { FontStyle, IThemedToken } from 'shiki'
3 import {
4   PDFDocument,
5   PDFFont,
6   PDFPageDrawTextOptions,
7   rgb,
8   StandardFonts,
9 } from 'pdf-lib'
10 import {
11   LineNumberTransformations,
12   PdfRendererOptions,
13   RenderToPdfOptions,
14 } from './types'
15 import { createPage, finishPage } from './page-utils'
16 import { chunkString, hexToRgb } from './utils'
17
18 const defaultColor = '#000000'
19
20 export const renderToPdf = async (
21   lines: IThemedToken[][],
22   pdfDocument: PDFDocument,
23   { fontMap, fontSize, bg, lineNumbers }: RenderToPdfOptions
24 ) => {
25   let { page, pageDimensions } = createPage(pdfDocument, bg)
26
27   const regularFont = fontMap.regular
28
29   const oneCharacterWidth = regularFont.widthOfTextAtSize('a', fontSize)
30   const maxCharactersPerLine = Math.floor(
31     pageDimensions.width / oneCharacterWidth
32   )
33   const largestLineNumberStringWidth = regularFont.widthOfTextAtSize(
34     lines.length.toString(),
35     fontSize
36   )
37
38   const startingLineX = largestLineNumberStringWidth + 10
39
40   let lineY = pageDimensions.height
41
42   let lineNumberTransformations: LineNumberTransformations = []
43
44   // eslint-disable-next-line @typescript-eslint/naming-convention
45   const subtractLineYByFontSize = () => {
46     lineY -= fontSize
47     if (lineY < 0) {
48       finishPage(page, startingLineX, lineNumberTransformations, lineNumbers)
49       lineNumberTransformations = []
50       ;({ page, pageDimensions } = createPage(pdfDocument, bg))
51       lineY = pageDimensions.height - fontSize
52     }
53   }
54
55   for (const [i, line] of lines.entries()) {
56     subtractLineYByFontSize()
57
58     const tokenText: string[] = []
59     let lineX = startingLineX
60     const lineNumberString = (i + 1).toString()
61
62     const currentLineY = lineY
63     lineNumberTransformations.push((currentPage) => {
64       currentPage.drawText(lineNumberString, {
65         x:
66           startingLineX -
67           regularFont.widthOfTextAtSize(lineNumberString, fontSize) -
68         5,

```

```

69     y: currentLineY,
70     size: fontSize,
71     color: lineNumbers.text,
72   })
73 })
74
75 for (const token of line) {
76   tokenText.push(token.content)
77
78   let tokenFont = fontMap.regular
79
80   if (token.fontStyle === FontStyle.Bold) {
81     tokenFont = fontMap.bold
82   } else if (token.fontStyle === FontStyle.Italic) {
83     tokenFont = fontMap.italic
84   }
85
86   const rgbColor = hexToRgb(token.color ?? defaultColor)
87   const tokenWidth = regularFont.widthOfTextAtSize(token.content, fontSize)
88
89   const drawOptions: PDFPageDrawTextOptions = {
90     size: fontSize,
91     color: rgbColor,
92     font: tokenFont,
93   }
94
95   if (tokenWidth > pageDimensions.width) {
96     const chunks = chunkString(token.content, maxCharactersPerLine)
97
98     for (const chunk of chunks) {
99       page.drawText(chunk, {
100         x: lineX,
101         y: lineY,
102         ...drawOptions,
103       })
104
105       subtractLineYByFontSize()
106     }
107
108     lineX += tokenFont.widthOfTextAtSize(
109       chunks[chunks.length - 1],
110       fontSize
111     )
112   } else {
113     const potentialNewLineX = lineX + tokenWidth
114
115     if (potentialNewLineX > pageDimensions.width) {
116       lineX = startingLineX
117       subtractLineYByFontSize()
118     }
119
120     page.drawText(token.content, {
121       x: lineX,
122       y: lineY,
123       ...drawOptions,
124     })
125
126     lineX += tokenWidth
127   }
128 }
129 }
130
131 finishPage(page, startingLineX, lineNumberTransformations, lineNumbers)
132
133 return pdfDocument
134 }
135
136 export const getPdfRenderer = (options: PdfRendererOptions = {}) => {
137   const bg = options.bg ?? rgb(1, 1, 1)

```

```
138
139 const fontMap = options.fontMap ?? {
140     regular: StandardFonts.Courier,
141     italic: StandardFonts.CourierOblique,
142     bold: StandardFonts.CourierBold,
143 }
144
145 const fontSize = options.fontSize ?? 12
146
147 const decimal247 = 247 / 255
148 const decimal153 = 153 / 255
149
150 const lineNumbers = options.lineNumbers ?? {
151     bg: rgb(decimal247, decimal247, decimal247),
152     text: rgb(decimal153, decimal153, decimal153),
153 }
154
155 return {
156     renderToPdf: async (lines: IThemedToken[][], pdfDocument: PDFDocument) =>
{
157         const embedFontMap: Record<string, PDFFont> = {}
158
159         await Promise.all(
160             Object.entries(fontMap).map(async ([variation, font]) => {
161                 const embedFont = await pdfDocument.embedFont(font)
162                 embedFontMap[variation] = embedFont
163             })
164         )
165
166         return renderToPdf(lines, pdfDocument, {
167             bg,
168             fontMap: embedFontMap,
169             fontSize,
170             lineNumbers,
171         })
172     },
173 }
174 }
175
176 export { hexToRgb }
177
```