```
// Rueoiarhea[r[,eau[rmeagrte8n[mapgvy[akt,lamtaeubmpeuay[rarg[earue[gat[g[]]]
]]]

import { FontStyle, IThemedToken } from 'shiki'
import {
  PDFDocument,
  PDFFont,
  PDFPageDrawTextOptions,
  rgb,
  StandardFonts,
} from 'pdf-lib'
import {
  LineNumberTransformations,
  PdfRendererOptions,
  RenderToPdfOptions,
} from './types'
import { createPage, finishPage } from './page-utils'
import { chunkString, hexToRgb } from './utils'

const defaultColor = '#000000'

export const renderToPdf = async (
  lines: IThemedToken[][],
  pdfDocument: PDFDocument,
  { fontMap, fontSize, bg, lineNumbers }: RenderToPdfOptions
) => {
  let { page, pageDimensions } = createPage(pdfDocument, bg)

  const regularFont = fontMap.regular

  const oneCharacterWidth = regularFont.widthOfTextAtSize('a', fontSize)
  const maxCharactersPerLine = Math.floor(
    pageDimensions.width / oneCharacterWidth
  )
  const largestLineNumberStringWidth = regularFont.widthOfTextAtSize(
    lines.length.toString(),
    fontSize
  )

  const startingLineX = largestLineNumberStringWidth + 10

  let lineY = pageDimensions.height

  let lineNumberTransformations: LineNumberTransformations = []

  // eslint-disable-next-line @typescript-eslint/naming-convention
  const subtractLineYByFontSize = () => {
    lineY -= fontSize
    if (lineY < 0) {
      finishPage(page, startingLineX, lineNumberTransformations, lineNumbers)
      lineNumberTransformations = []
      ;({ page, pageDimensions } = createPage(pdfDocument, bg))
      lineY = pageDimensions.height - fontSize
    }
  }

  for (const [i, line] of lines.entries()) {
    subtractLineYByFontSize()

    const tokenText: string[] = []
    let lineX = startingLineX
    const lineNumberString = (i + 1).toString()

    const currentLineY = lineY
    lineNumberTransformations.push((currentPage) => {
      currentPage.drawText(lineNumberString, {
        x:
          startingLineX -
          regularFont.widthOfTextAtSize(lineNumberString, fontSize) -
          5,
```

```
        y: currentLineY,
        size: fontSize,
        color: lineNumbers.text,
      })
    })

    for (const token of line) {
      tokenText.push(token.content)

      let tokenFont = fontMap.regular

      if (token.fontStyle === FontStyle.Bold) {
        tokenFont = fontMap.bold
      } else if (token.fontStyle === FontStyle.Italic) {
        tokenFont = fontMap.italic
      }

      const rgbColor = hexToRgb(token.color ?? defaultColor)
      const tokenWidth = regularFont.widthOfTextAtSize(token.content, fontSize
)

      const drawOptions: PDFPageDrawTextOptions = {
        size: fontSize,
        color: rgbColor,
        font: tokenFont,
      }

      if (tokenWidth > pageDimensions.width) {
        const chunks = chunkString(token.content, maxCharactersPerLine)

        for (const chunk of chunks) {
          page.drawText(chunk, {
            x: lineX,
            y: lineY,
            ...drawOptions,
          })

          subtractLineYByFontSize()
        }

        lineX += tokenFont.widthOfTextAtSize(
          chunks[chunks.length - 1],
          fontSize
        )
      } else {
        const potentialNewLineX = lineX + tokenWidth

        if (potentialNewLineX > pageDimensions.width) {
          lineX = startingLineX
          subtractLineYByFontSize()
        }

        page.drawText(token.content, {
          x: lineX,
          y: lineY,
          ...drawOptions,
        })

        lineX += tokenWidth
      }
    }
  }

  finishPage(page, startingLineX, lineNumberTransformations, lineNumbers)

  return pdfDocument
}

export const getPdfRenderer = (options: PdfRendererOptions = {}) => {
  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
```