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

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

const defaultColor = '#000000'

const setupDoc = async () => {
  const doc = await PDFDocument.create()
  doc.registerFontkit(fontkit)

  const font = await doc.embedFont(StandardFonts.Courier)

  return { doc, font }
}

export const renderToPdf = async (lines: IThemedToken[][]) => {
  const { doc, font } = await setupDoc()
  let { page, pageDimensions } = createPage(doc)

  const fontSize = 12
  const oneCharacterWidth = font.widthOfTextAtSize('a', fontSize)
  const maxCharactersPerLine = Math.floor(
    pageDimensions.width / oneCharacterWidth
  )
  const largestLineNumberStringWidth = font.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)
      lineNumberTransformations = []
      ;({ page, pageDimensions } = createPage(doc))
      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 -
          font.widthOfTextAtSize(lineNumberString, fontSize) -
```

```
          5,
        y: currentLineY,
        size: fontSize,
        color: rgb(...hexToRgb('#999')),
      })
    })

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

      const decimalRgbColor = hexToRgb(token.color ?? defaultColor)

      const tokenWidth = font.widthOfTextAtSize(token.content, fontSize)

      const drawOptions: PDFPageDrawTextOptions = {
        size: fontSize,
        color: rgb(...decimalRgbColor),
        font,
      }

      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 += font.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)

  return doc
}
```