```ts
import fs from 'node:fs/promises'
import path from 'node:path'
import process from 'node:process'
import { PDFDocument } from 'pdf-lib'
import { getHighlighter, Lang, Theme } from 'shiki'
import { getPdfRenderer } from 'shiki-renderer-pdf'
import { globby } from 'globby'
import filename2prism from 'filename2prism'
import isPathInside from 'is-path-inside'

export type StringToPdfOptions = {
  theme?: Theme
  lang?: Lang | undefined
}

/**
 * Convert a string to a PDF document.
 */
export const stringToPdf = async (
  code: string,
  options: StringToPdfOptions = {}
) => {
  const highlighter = await getHighlighter({
    theme: options.theme,
  })

  const pdfRenderer = getPdfRenderer()

  const tokens = highlighter.codeToThemedTokens(code, options.lang)
  const pdfDocument = await PDFDocument.create()

  await pdfRenderer.renderToPdf(tokens, pdfDocument)

  return pdfDocument
}

type CompileOnlyOptions = {
  rootDir?: string
  outDir?: string
  include: string[]
  exclude?: string[]
}

type CompileOptions = CompileOnlyOptions & {
  theme: StringToPdfOptions['theme']
}

type RequiredCompileOptions = {
  theme: Theme
} & Required<CompileOnlyOptions>

export const normalizeOptions = (
  options: CompileOptions
): RequiredCompileOptions => {
  if (!options.include) throw new Error('no files are included')
  const theme = options.theme ?? 'light-plus'
  const exclude = options.exclude ?? ['node_modules']
  const rootDir = options.rootDir ?? process.cwd()
  const outDir = options.outDir ?? 'pdfs'

  return { ...options, exclude, theme, rootDir, outDir }
}

/**
 * Compile all files specified to pdfs.
 * @returns The number of files compiled.
 */
export const compilePdfs = async (options: CompileOptions) => {
  const normalizedOptions = normalizeOptions(options)
  const filepaths = await globby(normalizedOptions.include, {
```

```
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment, @typescript-eslint/prefer-ts-expect-error
    // @ts-ignore tsc does not error here, but tsup does
    ignore: normalizedOptions.exclude,
  })

  await Promise.all(
    filepaths.map(async (filepath) => {
      if (!isPathInside(filepath, normalizedOptions.rootDir)) {
        throw new Error(
          `${filepath} is not inside rootDir ${normalizedOptions.rootDir}`
        )
      }

      const lang = filename2prism(filepath)[0]
      const code = await fs.readFile(filepath, 'utf8')

      let pdfDocument: PDFDocument
      try {
        pdfDocument = await stringToPdf(code, {
          // @ts-expect-error These should be compatible for the most part
          lang,
          ...normalizedOptions,
        })
      } catch (error: unknown) {
        throw new Error(

// eslint-disable-next-line @typescript-eslint/restrict-template-expressions
          `Failed to compile ${filepath}. Ensure your "include" is correct and that only latin characters are used if you are using the default Courier font: ${error}`
        )
      }

      const filepathRelativeToRootDir = path.relative(
        normalizedOptions.rootDir,
        filepath
      )

      const outFilepath = path.join(
        normalizedOptions.outDir,
        `${filepathRelativeToRootDir}.pdf`
      )

      const pdfDocumentBytes = await pdfDocument.save()
      await fs.mkdir(path.dirname(outFilepath), { recursive: true })
      return fs.writeFile(outFilepath, pdfDocumentBytes, 'binary')
    })
  )

  return { count: filepaths.length }
}

```