Code Documentation

Source: /Users/jacob/research/xprinto/src

Generated on: 4/15/2025, 9:48:12 PM

This document contains a formatted representation of the code with syntax highlighting and line numbers for easy reference. Navigate through the document using the table of contents (if available).

Table of Contents

Note: Page numbers reflect actual document pagination.

index.ts Language: TYPESCRIPT

```typescript
// Re-export necessary functions
export       { generatePdfFromPath }                          from      './pdf/genera
export       { readPath }               from      './file-reader'               ;
export       { highlightCode }                    from      './syntax/highlighter'
export       { log,         LogLevel        , setVerboseLogging }
```

file-reader.ts Language: TYPESCRIPT

```typescript
import fs from 'fs-extra';
import path from 'path';
import { glob } from 'glob'; // Updated import statemen
import { log, LogLevel } from './utils/logger'

// Interface for file content
export interface FileInfo {
    path: string;
    relativePath: string;
    content: string;
    extension: string;
}

// Function to read a single file
export async function readFile( span class="hljs-params
    const content = ?: string /span>): await fs. readFile Promise FileInfo
rootPath
    const relativePath = rootPath ? path. relati
    const extension = path. (filePath, filePath) : path.
basename extname (filePath);

    return {
        path: filePath,
    relativePath,
    content,
    extension
  };
}

// Function to read files from a directory recursively
export async function readDirectory ( span class="
  /span>): Promise FileInfo []> {
string
    // Get all files in the directory and subdirectories, updated to use
async/await with glob
    const files = await glob ( '**/*.*' , {
        cwd: dirPath,
        nodir: true,
        ignore: ['**/node_modules/**' , '**/.git/**'
  });
// Ignore node_modules
and .git

    log ( span class="hljs-string">`Found ${files.
  ` /span>, LogLevel . INFO );
${dirPath}
    // Read all files
    const fileInfos: FileInfo [] = [];
    for ( const file of files) {
        const filePath = path. join (dirPath, file);
        try {
            const fileInfo = await readFile (filePath,
      fileInfos. push (fileInfo);
    } catch (err) {
            log ( span class="hljs-string">`Error reading file
```

file-reader.ts Language: TYPESCRIPT

```
    span class="hljs-subst">${(err                                    as    Error     ).message}
    }                                                                  LogLevel        . ERROR
  }

    return       fileInfos;
}

// Function to read from either a file or directory
export       async       function          readPath        ( span class="hljs-param
    const      stats = Promise     await     FileInfo fs.    [] {   (inputPath);      /
   span>):

    if   (stats.        isFile      ()) {
        const      fileInfo =             await       readFile        (inputPath);
        return       [fileInfo];
  }     else      if   (stats.        isDirectory           ()) {
        return        readDirectory             (inputPath);
  }     else     {
        throw       new     Error     ( span class="hljs-string">`Invalid path:
  }                                                                          /span>);
   is neither a file nor a directory`
}
```

cli.ts Language: TYPESCRIPT

```typescript
#!/usr/bin/env node

import { Command } from 'commander';
import path from 'path';
import fs from 'fs-extra';
import { generatePdfFromPath } from './pdf/genera
import { log, LogLevel } from './utils/logger'

const program = new Command();

program
  .name( 'xprinto' )
  .description( 'Convert code to beautiful PDFs with syntax highlighti
  .version( '1.0.0' )
  .argument( <span class="hljs-string">'path>'
  .option( <span class="hljs-string">'-o, --output )'File or directory
path to convert'
  .option( <span class="hljs-string">'-t, --title './output.pdf' 'Output
path for the PDF'
  .option( <span class="hljs-string">'--theme 'Code Documentation' 'Th
the PDF document'
  .option( <span class="hljs-string">'--font-size 'github' 'Syntax)
highlighting theme'
  .option( '--line-numbers' '10' ) , 'Show line numbers' 'Font size
for code'
  .option( '--no-line-numbers' , 'Hide line numbers'
  .option( '-v, --verbose' , 'Enable verbose logging'
  .action( async ( inputPath : string , options) => {
    try {
      // Set log level based on verbose flag
      if (options. verbose ) {
        log ( 'Verbose mode enabled' , LogLevel
      }

      // Resolve input path
      const resolvedPath = path. resolve (input

      // Check if path exists
      if (!fs. existsSync (resolvedPath)) {
        log ( <span class="hljs-string">`Path does not exist:
        process. LogLevel exit . ERROR ( 1 ); );
/span>,
      }

      // Generate PDF
      log ( <span class="hljs-string">`Converting
      await LogLevel generatePdfFromPath INFO ); ( /
span>,
        resolvedPath,
        options. output ,
        {
          title : options. title ,
          theme : options. theme ,
          fontSize : parseInt (options. fontSize
```

cli.ts Language: TYPESCRIPT

```
                    showLineNumbers                : options.          lineNumbers
        }
      );

            log   ( span class="hljs-string">`PDF generated successfully:
    }       catch      (err) {        ` /span>,         LogLevel        . SUCCESS
  ${options.output}
            log   ( span class="hljs-string">`Error:
    process.     ).message}      exit    ( 1 )}       /span>`       /span>,         LogLevel
  as
  }
});

program.        parse     ();
```

utils/logger.ts Language: TYPESCRIPT

```typescript
export enum LogLevel {
  ERROR = 'ERROR',
  WARNING = 'WARNING',
  INFO = 'INFO',
  SUCCESS = 'SUCCESS',
  DEBUG = 'DEBUG'
}

const COLORS = {
  [LogLevel.ERROR]: '\x1b[31m',   // Red
  [LogLevel.WARNING]: '\x1b[33m',   // Yellow
  [LogLevel.INFO]: '\x1b[36m',   // Cyan
  [LogLevel.SUCCESS]: '\x1b[32m',   // Green
  [LogLevel.DEBUG]: '\x1b[35m',   // Magenta
  RESET: '\x1b[0m'   // Reset
};

let verboseLogging = false;

export function setVerboseLogging(verbose: boolean): void {
  verboseLogging = verbose;
}

export function log(level = LogLevel.INFO, message: string): void {
  // Only log DEBUG messages if verbose logging is enabled
  if (level === LogLevel.DEBUG && !verboseLogging) return;

  const timestamp = new Date().toISOString();
  const color = COLORS[level] || COLORS.INFO;

  console.log(`${color}[${timestamp}] [${level}] ${message}${COLORS.RESET}`);
}
```

syntax/highlighter.ts Language: TYPESCRIPT

```typescript
import hljs from 'highlight.js';
import { FileInfo } from '../file-reader';
import { log, LogLevel } from '../utils/logger';


// Language mapping for extensions not automatically recognized by highlight.js
const LANGUAGE_MAP: Record<string, string> = {
    'ts': 'typescript',
    'js': 'javascript',
    'jsx': 'javascript',
    'tsx': 'typescript',
    'md': 'markdown',
    'yml': 'yaml',
    // Add more mappings as needed
};

export interface HighlightedLine {
    line: string;
    lineNumber: number;
    tokens: {
        text: string;
        color?: string;
        fontStyle?: string;
    }[];
}

export interface HighlightedFile extends FileInfo {
    highlightedLines: HighlightedLine[];
    language: string;
}

// Function to get language for syntax highlighting
function getLanguage(extension: string): string {
    return LANGUAGE_MAP[extension.toLowerCase()] || 'plaintext';
}

// Function to decode HTML entities
function decodeHtmlEntities(text: string): string {
    const entities: Record<string, string> = {
        '&amp;': '&',
        '&lt;': '<',
        '&gt;': '>',
        '&quot;': '"',
        '&#39;': "'",
        '&apos;': "'",
        '&#x2F;': '/',
        '&#x60;': '`',
    };

```

syntax/highlighter.ts Language: TYPESCRIPT

```typescript
    // Replace all known entities
    return text.replace( /&|<|>|"|'|'|/|`|&#x[\dA-Fa-f]{2,};/g
    match => entities[match] || match);
}

// Function to highlight code based on file extension
export function highlightCode ( span class="hljs-params">
    const language = HighlightedFile getLanguage { (fileInfo. / exte
span>):

    try {
        // Split content into lines
        const lines = fileInfo. content . split ( '\n

        const highlightedLines : HighlightedLine
            let highlighted; line, index span) => span class="hljs
function">(

            // Try to highlight with specific language
            try {
        highlighted = hljs. highlight (line, { lang
      } catch (e) {
            // Fall back to auto detection
        highlighted = hljs. highlightAuto (lin
      }

            // Parse hljs output to extract tokens
            const tokens = parseHighlightedTokens

            return {
        line,
                lineNumber : index + 1 ,
        tokens
      };
    });

        return {
      ...fileInfo,
      highlightedLines,
      language
    };
  } catch (err) {
        log ( span class="hljs-string">`Error highlighting code for
                : span class="hljs-subst">${(err
${fileInfo.path} LogLevel . ERROR ); /span>`
span>,
        // Return basic line-by-line structure without highlighting
        const lines = fileInfo. content . split ( '\n
        const highlightedLines = lines. map ( span c
line, ) => /span> ({ line,
index
```

syntax/highlighter.ts Language: TYPESCRIPT

```
            lineNumber: index + 1,
            tokens: [{ text: line }]
        }));

        return {
          ...fileInfo,
          highlightedLines,
          language
        };
      }
    }

    // Function to parse hljs output into tokens
    function parseHighlightedTokens(span class="hljs-para
    /span>): { text: string; color?: str
    string
        const tokens: { text: string; color?: stri
    }[] {
        }[] =
        [];
        // Pre-process: decode HTML entities in the entire string
        const decodedHtml = decodeHtmlEntities(high

        // Regular expression to find span elements with class
        const regex = span class="hljs-regexp">/
        let match; ]+)/g /span>; \/
    span>|([^
        while ((match = regex.exec(decodedHtml)) !==
            if (match[3]) {
                // Plain text without span
                const text = decodeHtmlEntities(match[
              tokens.push({ text });
            } else {
                // Text with highlighting
                const className = match[1]; // hljs-keyword,
                const text = decodeHtmlEntities(match[

                // Map hljs classes to colors/styles
                const color = getColorForClass(className);
                const fontStyle = getFontStyleForClass

              tokens.push({ text, color, fontStyle });
            }
          }

        return tokens;
    }

    // Map hljs classes to colors (improved color scheme)
    function getColorForClass( span class="hljs-params">
      string | undefined { /
    span>):
```

syntax/highlighter.ts Language: TYPESCRIPT

```typescript
    if (className.includes('keyword')) return
    if (className.includes('string')) return
    if (className.includes('comment')) return
    if (className.includes('number')) return
    if (className.includes('function')) return
    if (className.includes('title')) return
    if (className.includes('params')) return
    if (className.includes('built_in')) return
    if (className.includes('literal')) return
    if (className.includes('property')) return
    if (className.includes('operator')) return
    if (className.includes('punctuation')) re
    return undefined;
}

// Map hljs classes to font styles
function getFontStyleForClass(span class="hljs-params"
    if (className.string | undefined.includes('comment')) return
    span>);
    if (className.includes('bold')) return 'b
    if (className.includes('italic')) return
    if (className.includes('emphasis')) return
    if (className.includes('strong')) return
    return undefined;
}
```

pdf/toc.ts Language: TYPESCRIPT

```typescript
import PDFDocument from 'pdfkit';
import { HighlightedFile } from '../syntax/highlighte
import { PdfOptions } from './generator';

export function generateTOC(
    doc: PDFKit.PDFDocument,
    files: HighlightedFile[],
    options: PdfOptions
): void {
    // Add a new page for TOC
  doc.addPage();

    // Set up page dimensions
    const pageWidth = options.paperSize[0] - (
                options.margins!.right;
    // Add TOC title
  doc.font('Helvetica-Bold')
     .fontSize(18)
     .text('Table of Contents', { align: 'c
     .moveDown(2);

    // Group files by directories for a hierarchical TOC
    const filesByDirectory: Record<string, High

  files.forEach(file => {
      const dirPath = file.relativePath.split
      if (!filesByDirectory[dirPath]) {
        filesByDirectory[dirPath] = [];
    }
    filesByDirectory[dirPath].push(file);
  });

    // Calculate actual page numbers
    // Cover page + TOC page = 2 pages before files
    let currentPage = 3;
    const pageNumbers: Record<string, number> =

    // Calculate page numbers first
    const directories = Object.keys(filesByDirectory)
  directories.forEach(directory => {
      const sortedFiles = filesByDirectory[directory].
    a.relativePath.localeCompare(b.relativePa
    );

    sortedFiles.forEach(file => {
      pageNumbers[file.relativePath] = currentPag

          // Calculate realistic page count based on file size
```

pdf/toc.ts Language: TYPESCRIPT

```
const lineCount = file.highlightedLines
const linesPerPage = Math.floor((options.
!. top - options. margins !. bottom - headerHeight
margins options.
const estimatedPages = fontSize * 1.4)); Math.max( 1 , Ma
!) / (options.
currentPage += estimatedPages; (lineCount /
linesPerPage));
});
});

// Now render the TOC with accurate page numbers
directories.forEach( span class="hljs-function">
if (directory) {
// Add directory name with better formatting
doc.font( 'Helvetica-Bold' )
.fontSize( 14 )
.fillColor( '#000000' );

// Add a box around the directory name
const directoryText = directory;
const textWidth = doc.widthOfString (d
const textHeight = doc.currentLineHeight

doc.rect(
options.margins !. left ,
doc.y ,
pageWidth,
textHeight + 8
)
.fillColor( '#f0f0f0' )
.fill();

// Write directory name
doc.fillColor( '#000000' )
.text(directoryText, options.margins
4 ); -
textHeight +
doc.moveDown( 1 );
}

// Sort files in the directory
const sortedFiles = filesByDirectory[directory].
a. relativePath .localeCompare (a, b) => /span span class="hljs- (b. relativePa
function">(
);

// Add file entries
sortedFiles.forEach( span class="hljs-function">
const fileName = file.relativePath .sp
const indent = directory ? ' ' :
relativePath
```

pdf/toc.ts Language: TYPESCRIPT

```
            // Get page number for this file
            const      pageNum = pageNumbers[file.                              relati

            // Calculate positions
            const      startX = options.                      margins       !.  left
            const      pageNumWidth = doc.                        widthOfString
            const      endX = options.                     margins       !.  left      + p
            const      nameWidth = endX - startX -                                  20

            // Add file name
      doc.           font     ( 'Helvetica'           )
         .           fontSize        ( 12  )
         .           fillColor         ( '#000000'          )
         .           text     ( span class="hljs-string">`                              ${
    startX, doc.             continued     y , {:   true    ,            /span>,
                             width     : nameWidth
        });

            // Create a dot leader that's more compact
            const      dotLeader =
      doc.           fillColor         ( '#888888'          )
    '. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .'
         .           text     (dotLeader, {                   continued         :   true

            // Add page number
      doc.           fillColor         ( '#000000'          )
         .           font     ( 'Helvetica-Bold'                )
         .           text     ( span class="hljs-string">`
                  });
    'right'
      doc.           moveDown        ( 0.5    );
    });

    doc.         moveDown        ( 0.5    );
  });

  // Add note about page numbers
  doc.       moveDown        ( 2 )
     .       font     ( 'Helvetica-Oblique'                   )
     .       fontSize        ( 10   )
     .       fillColor         ( '#555555'          )
     .       text     ( 'Note: Page numbers reflect actual document pagination.'
               align     :    'center'         ,
               width     : pageWidth
     });
}
```

pdf/page.ts Language: TYPESCRIPT

```typescript
import PDFDocument from 'pdfkit';
import { HighlightedFile } from '../syntax/highlight
import { PdfOptions } from './generator';

// Track page number across page renders - global counter
let currentPageNumber = 1;

export function renderPage(
  doc: PDFKit.PDFDocument,
  file: HighlightedFile,
  options: PdfOptions
): void {
  const pageWidth = options.paperSize[0] - options.margins!.right;
  const contentHeight = options.paperSize[1] - options.margins!.bottom - options.footerHeight - options.

  // Add header with file path
  renderHeader(doc, file, options);

  // Calculate starting position after header
  const startY = options.margins!.top + options.
  doc.y = startY;

  // Calculate line number column width based on the number of lines
  const maxLineNumber = file.highlightedLines.length;
  const lineNumberWidth = options.showLineNumbers * options.fontSize
  (maxLineNumber).
  // Render code
  renderCodeBlock(doc, file, options, lineNumberWidth, startY, contentHeight);
  // Add footer with page number
  renderFooter(doc, options, currentPageNumber);
}

function renderHeader(
  doc: PDFKit.PDFDocument,
  file: HighlightedFile,
  options: PdfOptions
): void {
  const headerY = options.margins!.top;
  const pageWidth = options.paperSize[0] - options.margins!.right - options.
  // Draw background for header
  doc.rect(options.margins!.left, headerY, pageWidth
  .fillColor('#f8f8f8')
  .fill();

```

pdf/page.ts Language: TYPESCRIPT

```
    // Draw file path
  doc. font ( 'Helvetica-Bold' )
    . fontSize ( 12 )
    . fillColor ( '#333333' )
    . text (file. relativePath , options. margins
      width : pageWidth - 150 ,
      align : 'left'
    });

    // Draw language
  doc. font ( 'Helvetica' )
    . fontSize ( 10 )
    . fillColor ( '#666666' )
    . text ( span class="hljs-string">`Language:
                                              ` /span>, options.
   ${file.language.toUpperCase()}
      width : headerY + 150 , 5 , { + pageWidth -
      align 140 : 'right'
    });

    // Draw a line under the header
  doc. moveTo (options. margins !. left , headerY + optio
    . lineTo (options. paperSize ![ 0 ] - options.
    . lineWidth ( headerHeight ! - , 1 ) headerY +
   options.
    . strokeColor ( '#dddddd' )
    . stroke ();
}

function renderFooter ( span class="hljs-params">
    doc : PDFKit . PDFDocument ,
    options : PdfOptions ,
    pageNumber : number
): void {
    const footerY = options. paperSize ![ 1 ] - optio
    const pageWidth = options. footerHeight !; paperSize - ![ 0 ] - o
   options.
              margins !. right ; -
   options.
    // Draw a line above the footer
  doc. moveTo (options. margins !. left , footerY)
    . lineTo (options. paperSize ![ 0 ] - options.
    . lineWidth ( 1 )
    . strokeColor ( '#dddddd' )
    . stroke ();

    // Draw page number using the global counter
  doc. font ( 'Helvetica' )
    . fontSize ( 10 )
    . fillColor ( '#666666' )
    . text ( span class="hljs-string">`Page
```

pdf/page.ts Language: TYPESCRIPT

```
    margins.left, footerY + 10, {
      width: pageWidth,
      align: 'center'
    });
}

function renderCodeBlock(
  doc: PDFKit.PDFDocument,
  file: HighlightedFile,
  options: PdfOptions,
  lineNumberWidth: number,
  startY: number,
  contentHeight: number
): void {
  // Set up monospaced font for code
  doc.font('Courier')
     .fontSize(options.fontSize);

  // Calculate available width for code
  const codeWidth = options.paperSize[0] -
    options.margins.left - options.margins.right - lineNumberWidth -
    10;

  // Current Y position for drawing
  let currentY = startY;
  const lineHeight = options.fontSize * 1.4; // Line height slightly
larger than font size

  // Draw background for the code block
  doc.rect(options.margins.left, startY, options.paperSize[0] -
    options.margins.left - options.margins.right, contentHeight)
     .fill('#f8f8f8');

  // Draw line number background if line numbers are shown
  if (options.showLineNumbers) {
    doc.rect(options.margins.left, startY, lineNumberWidth, contentHeight)
       .fill('#e8e8e8');
  }

  // Render each line
  for (let i = 0; i < file.highlightedLines.length; i++) {
    const line = file.highlightedLines[i];

    // Check if we need a new page
    if (currentY + lineHeight > startY + contentHeight) {
      // Add footer to current page
      renderFooter(doc, options, currentPageNumber);

      // Add a new page
      doc.addPage();

      // Increment page counter
```

```
      currentPageNumber++;

            // Reset current Y position
      currentY = startY;

            // Add header to new page
            renderHeader            (doc, file, options);

            // Draw background for the code block on the new page
      doc.          rect     (options.          margins       !.  left     , startY, op
options.margins!.left - options.margins
         .          fill     ( '#f8f8f8' );

            // Draw line number background if line numbers are shown
            if   (options.          showLineNumbers               ) {
        doc.            rect     (options.          margins       !.  left
contentHeight)
           .            fill     ( '#e8e8e8' startY, lineNumberWidth, );
      }
    }

        // Draw line number if enabled
        if   (options.          showLineNumbers               ) {
      doc.          font     ( 'Courier-Bold'              )
         .          fillColor         ( '#888888'          )
         .          text     (
                      String      (line.      lineNumber          ).  padStart
length     options.length      ,   ' ' ),margins       !.  left     +   5 ,
              currentY,
              {              lineBreak         :   false      }
         );
    }

        // Calculate starting X position for code
        const      codeX = options.                   margins       !.  left     + (op
lineNumberWidth +                           10   :    0 );              ?
        // Draw code with syntax highlighting
        let     currentX = codeX;
        let     lineWrapped =                false     ;

        for     (  const      token         of   line.      tokens       ) {
            // Set color for token
      doc.          font     (token.       fontStyle           ===      'bold'        ?
fontStyle
         .          fillColor === 'italic' (token.   ?   color 'Courier-Oblique' || '#000000'

            // Calculate width of token text
            const      tokenWidth = doc.                widthOfString

            // Check if token fits on current line
```

pdf/page.ts Language: TYPESCRIPT

```
            if (currentX + tokenWidth > codeX + codeWidth) {
                // Move to next line
        currentY += lineHeight;
        currentX = codeX +                          20  ;   // Indent continuation
        lineWrapped =                       true    ;

                // Check if we need a new page
                if  (currentY + lineHeight > startY + contentHeight) {
                    // Add footer to current page
                    renderFooter            (doc, options, currentPageNumber);

                    // Add a new page
          doc.              addPage       ();

                    // Increment page counter
          currentPageNumber++;

                    // Reset current Y position
          currentY = startY;

                    // Add header to new page
                    renderHeader            (doc, file, options);

                    // Draw background for the code block on the new page
          doc.              rect    (options.         margins       !.  left    , sta
            .           margins     fill    ( '#f8f8f8'    - options.          margins
    options.

                    // Draw line number background if line numbers are shown
                    if  (options.         showLineNumbers               ) {
            doc.                rect    (options.         margins       !.  left
              .                 fill    ( '#e8e8e8'       startY, lineNumberWidth,
    contentHeight)
          }
        }
      }

            // Draw token text
      doc.          text    (token.       text    , currentX, currentY, {

            // Update current X position
      currentX += tokenWidth;
    }

        // End the line
    doc.        text    ( ''  ,   0 ,   0 );

        // Move to next line (add extra space if line was wrapped)
    currentY += lineWrapped ? lineHeight *
  }
```

pdf/page.ts Language: TYPESCRIPT

```
235 }
```

pdf/generator.ts Language: TYPESCRIPT

```typescript
import PDFDocument from 'pdfkit';
import fs from 'fs-extra';
import path from 'path';
import { readPath } from '../file-reader';
import { highlightCode, HighlightedFile } from '../syntax-highlighter';
import { renderPage } from './page'; // Removed
import { generateTOC } from './toc';
import { log, LogLevel } from '../utils/logger';

export interface PdfOptions {
    title: string;
    theme: string;
    fontSize: number;
    showLineNumbers: boolean;
    paperSize?: [number, number]; // Width, height in
    margins?: { top: number; right: number;
    headerHeight?: number;
    footerHeight?: number;
}

// Default options
const defaultOptions: PdfOptions = {
    title: 'Code Documentation',
    theme: 'github',
    fontSize: 10,
    showLineNumbers: true,
    paperSize: [595.28, 841.89], // A4
    margins: { top: 50, right: 50, bottom: 50
    headerHeight: 30,
    footerHeight: 30
};

export async function generatePdfFromPath(
    inputPath: string,
    outputPath: string,
    options: Partial<PdfOptions> = {}
): Promise<void> {
    // Merge options with defaults
    const mergedOptions: PdfOptions = { ...defaultOpti

    try {
        // Read all files
        log(LogLevel.INFO, `Reading files from
    span>,
        const files = await readPath(inputPath);

        // Highlight code for all files
        log('Applying syntax highlighting...',
        const highlightedFiles: HighlightedFile
```

pdf/generator.ts Language: TYPESCRIPT

```
span class="hljs-function">                                    file     =>   /span>        highli
        // Create PDF document
        log   ( 'Generating PDF...'                    ,   LogLevel        . INFO
        const      doc =        new     PDFDocument           ({
            size    : mergedOptions.                paperSize         ,
            margins       : mergedOptions.                margins       ,
            info    : {
                Title     : mergedOptions.                title     ,
                Author      :   'Generated by xprinto'                          ,
                Creator       :   'xprinto'
          }
      });

        // Create write stream
        const      writeStream = fs.                createWriteStream
      doc.        pipe    (writeStream);

        // Add cover page
        addCoverPage            (doc, inputPath, mergedOptions);

        // Add table of contents if there are multiple files
        if   (highlightedFiles.                   length       >    1 ) {
            log   ( 'Generating table of contents...'
            generateTOC           (doc, highlightedFiles, mergedOptions);
      }

        // Add each file to the PDF
        for    (  const      file       of   highlightedFiles) {
            log   ( span class="hljs-string">`Adding file to PDF:
          doc.          addPage       ();       ` /span>,         LogLevel        . INFO
      ${file.relativePath}

            // Do NOT reset page counter before rendering each file
            // resetPageCounter() - removed this line

            renderPage          (doc, file, mergedOptions);
      }

        // Finalize the PDF
      doc.        end   ();

        // Wait for the write stream to finish
        await       new     Promise       void    >(  span class="hljs-function">(
          writeStream.                  on  ( 'finish'        ,   () =>      {   /
      span> {
                log   ( span class="hljs-string">`PDF written to
                resolve     . SUCCESS       );                          /span>,
            LogLevel
          });
          writeStream.                  on  ( 'error'       , reject);
```

pdf/generator.ts Language: TYPESCRIPT

```
    });
  } catch (err) {
    throw new Error(`Failed to generate PDF: ${(err as Error).message}`);
  }
}

// Function to add a cover page
function addCoverPage(doc: PDFKit.PDFDocument, inputPath: string, options: PdfOptions): void {
  // Set font for cover page
  doc.font('Helvetica-Bold')
     .fontSize(24)
     .text(options.title, { align: 'center' })
     .moveDown(2);

  // Add input path information
  doc.font('Helvetica')
     .fontSize(14)
     .text(`Source: ${inputPath}`, { align: 'center' })
     .moveDown(1);

  // Add generation date
  doc.fontSize(12)
     .text(`Generated on: ${new Date().toLocaleString()}`, { align: 'center' })
     .moveDown(4);

  // Add separator line
  const pageWidth = options.paperSize[0] - options.margins.left - options.margins.right;
  doc.moveTo(options.margins.left, doc.y)
     .lineTo(options.margins.left + pageWidth, doc.y)
     .stroke();

  // Add description
  doc.moveDown(2)
     .fontSize(12)
     .text(
       'This document contains a formatted representation of the code with syntax highlighting and line numbers for easy reference. Navigate through the document using the table of contents (if available).',
       {
         align: 'left',
         width: pageWidth
       }
     );
}
```