readme.md for @push.rocks/smartpdf

> **Transform HTML, websites, and PDFs into beautiful documents and images with just a few lines of code.**

npm version TypeScript License: MIT

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Why SmartPDF?

SmartPDF is your Swiss Army knife for PDF operations in Node.js. Whether you're generating invoices from HTML, snapshotting web pages, merging documents, or converting PDF pages to images — SmartPDF handles it all through a clean, async-first TypeScript API backed by headless Chromium.

□ Features at a Glance

Feature	Description
□□ **HTML → PDF**	Render any HTML string (with full CSS) into an A4-sized PDF
□□ **Website → PDF**	Capture a live URL as a PDF — either A4 or full-page scroll
□□ **PDF Merging**	Combine multiple PDF buffers into a single document

Feature	Description
□□ **PDF → Images**	Convert PDF pages to **PNG**, **WebP**, or progressive **JPEG**
□□ **Text Extraction**	Pull raw text content from any PDF buffer
□□ **Smart Port Management**	Automatic port allocation so multiple instances never collide
□□ **DPI Control**	Built-in scale constants for screen, high-quality, and print resolutions
□□ **BYO Browser**	Optionally pass your own Puppeteer `Browser` instance

□□ Installation

```
pnpm add @push.rocks/smartpdf
```

> **Prerequisites:** SmartPDF uses headless Chromium via Puppeteer under the hood. On most systems this is handled automatically. If you run into browser-launch issues (CI, Docker, etc.), make sure the required system libraries are installed — see the Puppeteer troubleshooting guide.

□□ Quick Start

```
import { SmartPdf } from '@push.rocks/smartpdf';
import * as fs from 'fs';

// 1. Create and start
const smartPdf = await SmartPdf.create();
await smartPdf.start();

// 2. Generate a PDF from HTML
const pdf = await smartPdf.getA4PdfResultForHtmlString(`
  <h1>Hello, PDF World! □□/h1>
  <p>Generated with SmartPDF.</p>
`);

// 3. Write to disk
```

```
fs.writeFileSync('my-first.pdf', pdf.buffer);

// 4. Clean up
await smartPdf.stop();
```

Every method returns an `IPdf` object:

```
interface IPdf {
  id: string | null;        // Unique identifier
  name: string;             // Filename
  buffer: Buffer;           // Raw PDF bytes
  metadata?: {
    textExtraction?: string; // Extracted text (when available)
  };
}
```

□□ How It Works

SmartPDF spins up a lightweight HTTP server (via `@push.rocks/smartserve`) bound to `localhost` and a headless Chromium browser. When you call a generation method:

1. Your HTML is registered internally and served at `http://localhost:{port}/{id}`
2. Puppeteer navigates to that URL, waits for the page to fully render, and captures a PDF
3. A header-based security check ensures only the correct content is captured
4. The server and browser are torn down when you call `stop()`

This architecture means you get **pixel-perfect CSS rendering**, **web font support**, and **full JavaScript execution** — the same rendering engine that powers Chrome.

□□ Instance Management

```
const smartPdf = await SmartPdf.create();
await smartPdf.start();

// ... your operations ...

await smartPdf.stop();
```

For production use, wrap in try/finally:

```
const smartPdf = await SmartPdf.create();
try {
  await smartPdf.start();
  // ... generate PDFs ...
} finally {
  await smartPdf.stop();
}
```

□□ Smart Port Allocation

Run multiple instances without conflicts:

```
// Each instance auto-selects a free port (default range: 20000–30000)
const instance1 = new SmartPdf();
const instance2 = new SmartPdf();
await instance1.start(); // e.g. port 20000
await instance2.start(); // e.g. port 20001

console.log(instance1.serverPort); // 20000
console.log(instance2.serverPort); // 20001

// Custom range
const custom = new SmartPdf({ portRangeStart: 4000, portRangeEnd: 5000 });

// Or pin a specific port
const pinned = new SmartPdf({ port: 3000 });
```

If a specific port is already in use, `start()` throws an error immediately instead of silently failing.

□□ Bring Your Own Browser

Pass an existing Puppeteer `Browser` instance — SmartPDF won't close it when you call `stop()`:

```
import puppeteer from 'puppeteer';

const browser = await puppeteer.launch({
```

```
  headless: 'new',
  args: ['--no-sandbox'],
});

const smartPdf = await SmartPdf.create();
await smartPdf.start(browser); // uses your browser

await smartPdf.stop();      // server stops, browser stays open
await browser.close();       // you manage browser lifecycle
```

□□ PDF Generation

□□ HTML → A4 PDF

Renders at a 794×1122 viewport (A4 at 96 DPI) with full CSS support:

```
const pdf = await smartPdf.getA4PdfResultForHtmlString(`
  <style>
    body { font-family: 'Helvetica', sans-serif; margin: 40px; }
    .header {
      background: linear-gradient(135deg, #667eea, #764ba2);
      color: white; padding: 30px; border-radius: 10px; text-align: center;
    }
    table { width: 100%; border-collapse: collapse; margin-top: 20px; }
    th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
    th { background: #f5f5f5; }
  </style>

  <div class="header">
    <h1>Invoice #2024-001</h1>
  </div>

  <table>
    <tr><th>Item</th><th>Qty</th><th>Price</th></tr>
    <tr><td>Widget Pro</td><td>5</td><td>$49.99</td></tr>
    <tr><td>Gizmo Ultra</td><td>2</td><td>$129.99</td></tr>
  </table>
```

```
`);

fs.writeFileSync('invoice.pdf', pdf.buffer);
```

□□ Website → PDF

Two methods depending on your needs:

```
// Standard capture — uses the document's own dimensions
const pdf = await smartPdf.getPdfResultForWebsite('https://example.com');

// Full-page capture — scrolls to bottom, captures everything as a single page
const fullPdf = await smartPdf.getFullWebsiteAsSinglePdf('https://example.com');
```

`getPdfResultForWebsite` uses a 1980×1200 viewport and respects the page's own width/height. `getFullWebsiteAsSinglePdf` uses a 1920px-wide viewport and measures the full scroll height, producing a single tall page.

□□ Merge Multiple PDFs

Combine any number of PDF buffers into one document using `pdf-lib`:

```
const invoice = await smartPdf.readFileToPdfObject('./invoice.pdf');
const terms = await smartPdf.readFileToPdfObject('./terms.pdf');
const appendix = await smartPdf.getA4PdfResultForHtmlString('<h1>Appendix</h1>...');

const merged = await smartPdf.mergePdfs([
  invoice.buffer,
  terms.buffer,
  appendix.buffer,
]);

fs.writeFileSync('complete-package.pdf', merged);
```

□□ Read a PDF from Disk

```
const pdfObject = await smartPdf.readFileToPdfObject('./document.pdf');
console.log(pdfObject.name);   // "document.pdf"
```

```
console.log(pdfObject.buffer); // <Buffer ...>
```

□□ Extract Text

Pull raw text from any PDF buffer:

```
const text = await smartPdf.extractTextFromPdfBuffer(pdf.buffer);
console.log(text);
```

> Uses pdf2json under the hood. Works best with text-based PDFs; scanned documents may return limited results.

□□ PDF → Image Conversion

Convert PDF pages to raster images using Puppeteer + PDF.js. Each page becomes a separate image buffer.

PNG — Lossless Quality

```
const pngPages = await smartPdf.convertPDFToPngBytes(pdf.buffer, {
  scale: SmartPdf.SCALE_HIGH, // 3.0 = ~216 DPI (default)
});

pngPages.forEach((png, i) => {
  fs.writeFileSync(`page-${i + 1}.png`, Buffer.from(png));
});
```

WebP — Modern & Efficient

25–60% smaller than PNG at similar visual quality:

```
const webpPages = await smartPdf.convertPDFToWebpBytes(pdf.buffer, {
  scale: 2.0,     // ~144 DPI
  quality: 90,    // 0–100 (default: 85)
```

```
});
```

JPEG — Progressive Loading

Generates true progressive JPEGs (multi-pass rendering) via sharp:

```
const jpegPages = await smartPdf.convertPDFToJpegBytes(pdf.buffer, {
  scale: SmartPdf.SCALE_HIGH,
  quality: 85,       // 0–100 (default: 85)
  maxWidth: 1920,    // optional dimension constraints
  maxHeight: 1080,
});
```

□□ DPI & Scale Reference

All image methods accept a `scale` parameter. PDF.js renders at 72 DPI by default, so `scale` is a multiplier:

Constant	Value	DPI	Use Case
`SmartPdf.SCALE_SCREEN`	2.0	~144	Web display, thumbnails
`SmartPdf.SCALE_HIGH`	3.0	~216	General purpose (default)
`SmartPdf.SCALE_PRINT`	6.0	~432	Print-quality output

Or calculate a custom scale:

```
const scale = SmartPdf.getScaleForDPI(300); // → 4.167
```

□□ Dimension Constraints

All image methods support `maxWidth` and `maxHeight` to cap output size while preserving aspect ratio:

```
// High-res render, but capped at 800×1000 px
const constrained = await smartPdf.convertPDFToWebpBytes(pdf.buffer, {
  scale: SmartPdf.SCALE_HIGH,
  quality: 90,
  maxWidth: 800,
```

```
  maxHeight: 1000,
});
```

□□ Format Comparison

Format	Typical Size vs PNG	Lossy?	Transparency	Progressive	Best For
PNG	baseline	No	□	—	Screenshots, diagrams, text-heavy docs
WebP	40–75%	Yes	□	—	Modern web apps, thumbnails
JPEG	50–70%	Yes	□	□	Photos, complex graphics, email

⚡ Parallel Processing

Process multiple URLs concurrently with separate instances:

```
const urls = [
  'https://example.com/page1',
  'https://example.com/page2',
  'https://example.com/page3',
];

// Spin up parallel instances
const instances = await Promise.all(
  urls.map(() => SmartPdf.create())
);
await Promise.all(instances.map(i => i.start()));

// Generate in parallel
const pdfs = await Promise.all(
  urls.map((url, i) => instances[i].getFullWebsiteAsSinglePdf(url))
);

// Merge all results
```

```
const merged = await instances[0].mergePdfs(pdfs.map(p => p.buffer));
fs.writeFileSync('all-pages.pdf', merged);

// Clean up
await Promise.all(instances.map(i => i.stop()));
```

□□ Full API Reference

`SmartPdf` Class

Static Properties

Property	Type	Value	Description
`SCALE_SCREEN`	`number`	`2.0`	~144 DPI scale factor
`SCALE_HIGH`	`number`	`3.0`	~216 DPI scale factor (default)
`SCALE_PRINT`	`number`	`6.0`	~432 DPI scale factor

Static Methods

Method	Returns	Description
`create(options?)`	`Promise<SmartPdf>`	Factory method to create an instance
`getScaleForDPI(dpi)`	`number`	Converts a DPI value to a scale factor (`dpi / 72`)

Instance Properties

Property	Type	Description
`serverPort`	`number`	The port the internal HTTP server is listening on

Instance Methods

Method	Returns	Description
`start(browser?)`	`Promise<void>`	Starts internal server + browser. Optionally accepts an existing Puppeteer `Browser`.

Method	Returns	Description
`stop()`	`Promise<void>`	Shuts down server and browser (unless external browser was provided).
`getA4PdfResultForHtmlString(html)`	`Promise<IPdf>`	Renders HTML at 794×1122 viewport → A4 PDF
`getPdfResultForWebsite(url)`	`Promise<IPdf>`	Captures website at 1980×1200 viewport → PDF
`getFullWebsiteAsSinglePdf(url)`	`Promise<IPdf>`	Captures full scrollable page at 1920px wide → single-page PDF
`mergePdfs(buffers)`	`Promise<Uint8Array>`	Merges an array of PDF `Uint8Array` buffers
`readFileToPdfObject(path)`	`Promise<IPdf>`	Reads a PDF file from disk into an `IPdf` object
`extractTextFromPdfBuffer(buffer)`	`Promise<string>`	Extracts raw text from a PDF buffer
`convertPDFToPngBytes(buffer, opts?)`	`Promise<Uint8Array[]>`	Converts each PDF page to a PNG buffer
`convertPDFToWebpBytes(buffer, opts?)`	`Promise<Uint8Array[]>`	Converts each PDF page to a WebP buffer
`convertPDFToJpegBytes(buffer, opts?)`	`Promise<Uint8Array[]>`	Converts each PDF page to a progressive JPEG buffer

Image Conversion Options

```
{
  scale?: number;      // DPI multiplier (default: 3.0)
  quality?: number;    // 0–100, WebP/JPEG only (default: 85)
  maxWidth?: number;   // Max output width in pixels
  maxHeight?: number;  // Max output height in pixels
}
```

`ISmartPdfOptions` Interface

```
{
  port?: number;           // Use a specific port
  portRangeStart?: number; // Auto-allocation range start (default: 20000)
  portRangeEnd?: number;   // Auto-allocation range end (default: 30000)
}
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:11:38 UTC by foss.global Team
Updated 2026-03-28 12:18:30 UTC by foss.global Team