

# @push.rocks/smartpdf

A library for creating PDFs dynamically from HTML or websites with additional features like merging PDFs.

- [readme.md for @push.rocks/smartpdf](#)
- [changelog.md for @push.rocks/smartpdf](#)

# 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/](https://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/](https://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
☐ <b>HTML → PDF</b>	Render any HTML string (with full CSS) into an A4-sized PDF
☐ <b>Website → PDF</b>	Capture a live URL as a PDF — either A4 or full-page scroll
☐ <b>PDF Merging</b>	Combine multiple PDF buffers into a single document

Feature	Description
☐ PDF → Images	Convert PDF pages to <b>PNG</b> , <b>WebP</b> , or progressive <b>JPEG</b>
☐ 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 <code>Browser</code> 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/smartserv`) 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
<code>SmartPdf.SCALE_SCREEN</code>	2.0	~144	Web display, thumbnails
<code>SmartPdf.SCALE_HIGH</code>	3.0	~216	General purpose (default)
<code>SmartPdf.SCALE_PRINT</code>	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
<b>PNG</b>	baseline	No	☐	—	Screenshots, diagrams, text-heavy docs
<b>WebP</b>	40-75%	Yes	☐	—	Modern web apps, thumbnails
<b>JPEG</b>	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(pdfjs.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 ( $\text{dpi} / 72$ )

### 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 <code>Browser</code> .

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

# changelog.md for @push.rocks/smartpdf

## 2026-03-09 - 4.2.0 - feat(smartpdf)

replace internal Express server with @push.rocks/smartserv, add PDF→WebP rendering, improve start/stop handling and bump dependencies

- Replace internal Express HTTP implementation with @push.rocks/smartserv and update README wording to reflect HTTP server usage
- Add PDF→WebP rendering: use pdf.js in-page rendering, capture canvases via Puppeteer to produce WebP buffers; added robust wait/timeout and error handling
- Add start/stop guards: \_isRunning flag, reset readiness Deferred on start, and throw if start called while running
- Remove direct http/express exports from plugins and stop exporting express; export smartserv from plugins
- Improve JPEG conversion to produce progressive JPEGs via SmartJimp (sharp mode)
- Bump dependencies/devDependencies: @push.rocks/smartsfs to ^1.5.0, add @push.rocks/smartserv ^2.0.1; devDeps @git.zone/tsbuild ^4.3.0, @git.zone/tstest ^3.3.0, @types/node ^25.3.5

## 2026-03-01 - 4.1.3 - fix(tests)

use example.com in image conversion test and relax JPEG size assertion

- Replaced https://www.wikipedia.org with https://example.com in test/test.ts for the third PDF generation test
- Removed the strict expectation that JPEG size must be smaller than PNG; now only asserts that WebP is smaller than PNG
- Updated test comment to note that JPEG may not be smaller for simple graphics pages

## 2026-03-01 - 4.1.2 - fix(smartsfs)

replace smartfile with smartfs, update file reading to use SmartFs, remove GraphicsMagick/Ghostscript dependency checks, bump dev and runtime dependencies, update tests and docs, and adjust npmextra configuration

- Replace usage/export of @push.rocks/smartfile with @push.rocks/smartfs and update readfileToPdfObject to use SmartFs + SmartFsProviderNode
- Remove execFile import and the GraphicsMagick/Ghostscript dependency-checking helpers from smartpdf (no more gm/gs checks)
- Bump devDependencies: @git.zone/tsbuild ^4.1.2, @git.zone/tsdoc ^1.12.0, @git.zone/tsrun ^2.0.1, @git.zone/tstest ^3.1.8, @types/node ^25.3.2
- Bump runtime dependencies: @push.rocks/smartfs ^1.3.1, @push.rocks/smartnetwork ^4.4.0, @tsclass/tsclass ^9.3.0, @types/express ^5.0.6, express ^5.2.1, pdf2json ^4.0.2
- Tests updated: switched example URLs to example.com, added a third PDF generation test that writes .nogit/3.pdf, and exported tap.start() as default
- npmextra.json reorganized to namespaced keys, added release.registries and accessLevel, and adjusted tsdoc/legal entries
- Documentation/readme refreshed: added issue reporting/security section, feature table, and various wording/formatting updates

## 2025-08-02 - 4.1.0 - feat(smartpdf)

Add progressive JPEG generation support

- Added new convertPDFToJpegBytes method for progressive JPEG generation
- Integrated @push.rocks/smartjimp for true progressive JPEG encoding
- Progressive JPEGs load in multiple passes, showing low-quality preview first
- Supports quality and scale options like other image generation methods
- Updated readme with comprehensive documentation and modern styling
- Updated legal section to reflect Task Venture Capital GmbH ownership

## 2025-08-02 - 4.0.0 - BREAKING CHANGE(smartpdf)

Improve image generation quality and API consistency

- BREAKING: Renamed `convertPDFToWebpPreviews` to `convertPDFToWebpBytes` for API consistency
- Added configurable scale options to `convertPDFToPngBytes` method
- Changed default scale from 1.0 to 3.0 for PNG generation (216 DPI)

- Changed default scale from 0.5 to 3.0 for WebP generation (216 DPI)
- Added DPI helper methods: `getScaleForDPI()` and scale constants (`SCALE_SCREEN`, `SCALE_HIGH`, `SCALE_PRINT`)
- Added `maxWidth`/`maxHeight` constraints for both PNG and WebP generation
- Improved test file organization with clear naming conventions
- Updated documentation with DPI/scale guidance and examples

## 2025-08-01 - 3.3.0 - feat(smartpdf)

Add automatic port allocation and multi-instance support

- Added `ISmartPdfOptions` interface with port configuration options
- Implemented automatic port allocation between 20000-30000 by default
- Added support for custom port ranges via `portRangeStart`/`portRangeEnd` options
- Added support for specific port assignment via `port` option
- Fixed resource cleanup when port allocation fails
- Multiple `SmartPdf` instances can now run simultaneously without port conflicts
- Updated readme with comprehensive documentation for all features

## 2025-02-25 - 3.2.2 - fix(SmartPdf)

Fix buffer handling for PDF conversion and text extraction

- Ensure `Uint8Array` is converted to Node Buffer for PDF conversion.
- Correct the PDF page viewport handling by using document dimensions.
- Fix `extractTextFromPdfBuffer` argument type from `Uint8Array` to `Buffer`.

## 2025-02-25 - 3.2.1 - fix(SmartPdf)

Fix type for `extractTextFromPdfBuffer` function

- Corrected the parameter type from `Buffer` to `Uint8Array` for `extractTextFromPdfBuffer` function.

## 2025-02-25 - 3.2.0 - feat(smartpdf)

Improve dependency versions and optimize PDF to PNG conversion.

- Update several dependencies to newer versions for better stability and performance.
- Refactor tests to enhance readability and add directory creation validations.
- Optimize PDF to PNG conversion by switching to a more efficient Puppeteer and PDF.js-based method.
- Add checks for presence of required dependencies (GraphicsMagick and Ghostscript).
- Fix media emulation issue by properly awaiting the emulateMediaType function.

## 2024-11-30 - 3.1.8 - fix(core)

Fix candidate handling in PDF generation

- Added error handling for missing PDF candidates in server requests.
- Updated devDependencies and dependencies to latest versions for better stability and new features.
- Patched header retrieval logic during PDF generation for security check.

## 2024-09-27 - 3.1.7 - fix(dependencies)

Update dependencies to latest versions

- Updated @git.zone/tsbuild to version ^2.1.84
- Updated @git.zone/tsdoc to version ^1.3.12
- Updated @git.zone/tsrun to version ^1.2.49
- Updated @push.rocks/tapbundle to version ^5.3.0
- Updated @types/node to version ^22.7.4
- Updated @push.rocks/smartfile to version ^11.0.21
- Updated @push.rocks/smartpromise to version ^4.0.4
- Updated @tsclass/tsclass to version ^4.1.2
- Updated express to version ^4.21.0
- Updated pdf2pic to version ^3.1.3

## 2024-05-29 - 3.1.6 - Core

Updated description

- Minor changes to documentation and internal text.

# 2024-04-25 to 2024-04-30 - 3.1.0 to 3.1.5 - Core

Fix updates in core functionality

- Fixes and updates in core function in versions 3.1.0 to 3.1.5.

# 2024-04-25 - 3.0.17 - Feature

Now supports PDF to JPG conversion

- Added support for converting PDF files to JPG format.

# 2024-03-19 to 2024-04-14 - 3.0.17 - Maintenance

Various updates to project configuration files

- Updated `tsconfig`.
- Updated `npmextra.json`.

# 2023-07-11 to 2024-03-19 - 3.0.15 to 3.0.16 - Organization

Switch to new organization scheme and core updates

- Switched to new organization scheme.
- Applied core updates and bug fixes.

# 2022-11-07 to 2023-07-10 - 3.0.13 to 3.0.14 - Core

Fixes and updates to core functionality

- Various minor bug fixes and updates to core components.

# 2022-09-13 to 2022-11-07 - 3.0.10 to 3.0.12 - Core

Ongoing core updates and maintenance

- Regular fixes and operational improvements in core functionalities.

# 2022-06-12 to 2022-09-13 - 3.0.7 to 3.0.9 - Core

Continued focus on high-priority bug fixes and core functionalities

- Regular fixes for critical bugs and enhancements.

# 2022-03-24 to 2022-06-29 - 3.0.3 to 3.0.6 - Core

Further optimization and maintenance releases

- Further improvements and refinements of issues in core functionalities.

# 2022-01-05 to 2022-03-25 - 3.0.0 to 3.0.2 - Major Version Release

Major release for version 3.0.x, including core fixes

- Increased version from 2.x to 3.0. New significant changes and fixes.

# 2022-01-05 to 2022-03-24 - 2.0.13 to 2.0.19 - Core

Routine core updates and bug fixes

- Regular bug fixes in core components.

# 2019-11-19 to 2022-01-06 - 2.0.0 to 2.0.11 - Core

Multiple core updates and a few performance improvements

- Some performance enhancements and multiple bug fixes.

# 2019-11-16 to 2019-11-19 - 1.0.27 to 1.0.29 - API

Breaking change in API

- Naming PDF results to better represent their content.

# 2019-05-29 to 2019-11-15 - 1.0.13 to 1.0.26 - Core

Core functional updates and some major restructuring

- Introduced multiple updates to the core, addressing bugs and improving stability.

# 2019-04-10 to 2019-05-28 - 1.0.4 to 1.0.12 - Core

Fixes and updates in the core

- Implementation of multiple essential fixes for core components.

# 2018-10-06 - 1.0.1 to 1.0.3 - Core and Typings

Initial implementation and core fixes

- Initial implementation of the project.
- Fixed compilation problems in typings.

# 2016-01-29 - unknown - Initial

Initial commit

- Initial commit for the project setup.