Offline-Lesen

Blog-Artikel von HTML nach ePub und PDF exportieren

Willem L. Middelkoop
19 Jan. 2026

Dieses Jahr markiert das 10-jährige Jubiläum meines Blogs. Was als Experiment begann, hat sich zu über 170 Geschichten entwickelt, mit durchschnittlich rund 60.000 Pageviews pro Monat. Ich habe kürzlich angefangen, Beiträge ins Niederländische, Deutsche und Spanische zu übersetzen, um die Reichweite zu erweitern. Und jetzt führe ich Unterstützung für Offline-Lesen ein, indem ich alle Beiträge als ePub- und PDF-Downloads verfügbar mache.

Lies Beiträge von willem.com auf deinem Lieblings-E-Reader (wie Kobo oder reMarkable)

Warum offline?

Genieße ablenkungsfreies Lesen auf E-Ink-Geräten wie einem Kobo oder reMarkable – diese Bildschirme sind sanft zu deinen Augen. Konzentriere dich ganz auf den Artikel, wo und wann immer du möchtest, denn es handelt sich um Downloads, die komplett offline funktionieren und DRM-frei sind. Alle ePubs und PDFs von willem.com enthalten hochwertige Bilder und wunderschöne, skalierbare Typografie. Ich habe hier ein praktisches Übersichtsverzeichnis aller ePub- und PDF-Downloads zusammengestellt:

ePub und PDF herunterladen

Wie es funktioniert

Die Website willem.com läuft auf dem Lemmid-Content-Management-System, einem Static-Site-Generator, der hochoptimierte HTML-Dateien erzeugt – ideal für hohes Traffic-Aufkommen und internationale Verteilung. Ich habe den NodeJS-Code erweitert, um automatisch aus vereinfachtem HTML heraus ePub- und PDF-Versionen zu exportieren. Siehe die Code-Snippets unten: für **ePub** nutze ich das epub-gen-Modul; für **PDF** verwende ich Pandoc und LaTeX. Ich habe einen einfachen Task-Manager gebaut, der Ausgabe-Jobs für einzelne Übersetzungen und Dateiformate in die Warteschlange stellt und die Rechenlast gleichmäßig über das Backend verteilt.

Fazit

Die Reaktionen, die ich auf meine Beiträge bekomme, kommen aus der ganzen Welt und motivieren mich, weiterzuschreiben. Mit der ePub- und PDF-Unterstützung hoffe ich, die Reichweite noch weiter auszubauen, indem ich mehr Wege ermögliche, die Artikel zu lesen. Viel Spaß dabei!

```
Publisher.prototype.performOutputJobEpub = function(job, callback) {
   // console.log('Publisher: performOutputJobEpub '+job.instance.id+' as '+job.type);
   var instance = job.instance;
   var language = job.language;
   var languageSuffix = language.toUpperCase();
   var epubFilename = instance['slug' + languageSuffix] + '.epub';
   this.base.isFileNewerThanTimestamp(instance['absolutePath' + languageSuffix] + epubFilename, instance.lastUpdated, function(isNewer) {
      if (isNewer) {
         callback(); // no action needed
      } else {
         // the ePub needs to be generated
         const heroImagePath = (instance['image' + languageSuffix])
            ? this.base.resultPath + instance['image' + languageSuffix].url.w1000
            : this.base.resultPath + '/global/icon.png';
         // Localised strings for the Table of Contents
         const tocTitles = {
            en: "Table of Contents",
            nl: "Inhoudsopgave",
            de: "Inhaltsverzeichnis",
            es: "Tabla de Contenidos"
         };
         const fullTitle = instance['title' + languageSuffix] + ': ' + instance['subTitle' + languageSuffix];
         // the inputHtml is based on feedHtml which uses absolute URL's, since
         // we're processing locally on the filesystem into epubs, we're removing
         // online (image) links, transforming them in absolute filesystem paths,
         // we also remove the anti cache parameter ?=LASTUPDATED
         const inputHtml = '<h4>' + instance['dateFormatted' + languageSuffix] + '</h4>\n' +
               this.base.replaceAllInString(
                 this.base.replaceAllInString(instance['feedHtml' + languageSuffix],
                     'https://' + this.base.rootFolderName + instance['relativePath' + languageSuffix],
                     instance['absolutePath' + languageSuffix]),
                 '?u=' + instance.lastUpdated, '');

         const options = {
            title: fullTitle,
            author: this.authorName,
            publisher: this.base.rootFolderName,
            lang: language,
            cover: heroImagePath,
            tocTitle: tocTitles[language] || tocTitles['en'], tempDir: this.rootPath + '/tmp',
            content: this.base.splitHtmlIntoChapters(inputHtml, fullTitle),
            appendChapterTitles: true,
            verbose: this.isDebug
         };
         const outputPath = instance['absolutePath' + languageSuffix] + epubFilename;
         // Generate the EPUB using promise .then/.catch for callback hook.
         new Epub(options, outputPath).promise .then(() => {
            console.log(`Publisher: Successfully generated EPUB for ${instance.id}`);
            if (callback)
               callback();
         }) . catch ((error) => {
            console.error(`Publisher: Error generating EPUB for ${instance.id}:`, error); if (callback)
               callback();
         });
      } // !isNewer
   }
   .bind(this)); // executeIfPathIsOlder
}
```

ePub code mit epub-gen

```
Publisher.prototype.performOutputJobPdf = function(job, callback) {
   // console.log('Publisher: performOutputJobPdf '+job.instance.id+' as '+job.type);
   var instance = job.instance;
   var language = job.language;
   var languageSuffix = language.toUpperCase();
   var pdfFilename = instance['slug' + languageSuffix] + '.pdf';
   this.base.isFileNewerThanTimestamp(instance['absolutePath' + languageSuffix] + pdfFilename,
                                      instance.lastUpdated, function(isNewer) {
      if (isNewer) {
         callback();
         // no action needed
      } else {
         // the PDF needs to be generated
         console.log('Publisher: generating PDF for ' + job.instance.id);
         var inputHtml = `<html lang="${language}">
                          <head>
                              <title>${instance["title"+languageSuffix]}</title>
                          </head>
                          <body>
                              ${instance["feedHtml"+languageSuffix]}
                          </body>
                      </html>`;
         // the inputHtml is based on feedHtml which uses absolute URL's, since
         // we're processing locally on the filesystem into PDF's, we're removing
         // online (image) links, transforming them in absolute filesystem paths:
         // we also remove the anti cache parameter ?=LASTUPDATED
         inputHtml = this.base.replaceAllInString(this.base.replaceAllInString(inputHtml,
                        'https://' + this.base.rootFolderName + instance['relativePath' + languageSuffix],
                        instance['absolutePath' + languageSuffix]), '?u=' + instance.lastUpdated, '');
         // we can only do the PDF generation if we have the basic html as input,
         // so we make sure that file is written first:
         this.base.makeSureFileIsWritten(instance['absolutePath' + languageSuffix] + 'tmp-pdf-input.html', inputHtml, function() {
            // Then we compose variables used to start a pandoc process to generate a PDF
            // using LateX magic and the basic.html as input:
            const languageSuffix = language.toUpperCase();
            const inputPath = instance['absolutePath' + languageSuffix] + 'tmp-pdf-input.html';
            const outputPath = instance['absolutePath' + languageSuffix] + pdfFilename;
            const templatePath = this.base.rootPath + '/assets/template.latex';
            const title = this.base.escapeLatex(instance['title' + languageSuffix]);
            const subTitle = this.base.escapeLatex(instance['subTitle' + languageSuffix]);
            const author = this.base.escapeLatex(this.authorName);
            const date = instance["dateFormatted" + languageSuffix];
            const heroImagePath = (instance['image' + languageSuffix])
                ? this.base.resultPath + instance['image' + languageSuffix].url.w1000
                : this.base.resultPath + '/global/icon.png';
            // Construct the Pandoc command with the corrected engine
            const command = `pandoc \\
                             -f html \\
                             -o "${outputPath}" \\
                             --pdf-engine=xelatex \\
                             -V title="${title}" \\
                             -V subTitle="${subTitle}" \\
                             -V heroImagePath="${heroImagePath}" \\
                             -V author="${author}" \\
                             -V date="${date}" \\
                             --template="${templatePath}" \\
                             "${inputPath}"`;
            // Execute the command (and pray for some good luck)
            exec(command, (error, stdout, stderr) => {
               if (error) {
                  console.error(`Publisher: Error generating PDF for ${instance.id}: ${error.message}`);
                  callback();
                  // keep continuing on the other jobs
                  return;
               }
               // if (stderr) {
               // console.warn(`Publisher: Pandoc stderr for ${instance.id}: ${stderr}`);
               // }
               console.log(`Publisher: Successfully generated PDF at ${pdfFilename}`);
               // clear tmp input HTML file:
               fs.unlink(inputPath, function(error) {
                  if (error)
                     throw error;
                  console.log('Publisher: Removed tmp HTML file from ' + inputPath);
               });
               callback();
               // tell the job queue we're done by calling back
            });
            // exec pandoc
         }
         .bind(this)); // makeSureFileIsWritten basic.html
      } // !isNewer
   }
   .bind(this)); // executeIfPathIsOlder
}
```

PDF code mit Pandoc und LaTeX