Offline Lezen

Blogposts exporteren van HTML naar ePub en PDF

Willem L. Middelkoop
19 jan. 2026

Dit jaar markeert het 10-jarig jubileum van mijn blog. Wat begon als een experiment is uitgegroeid tot meer dan 170 verhalen, met gemiddeld zo'n 60 duizend pageviews per maand. Ik ben onlangs begonnen met het vertalen van posts naar het Nederlands, Duits en Spaans om het bereik te vergroten. En nu introduceer ik ondersteuning voor offline lezen door alle posts beschikbaar te maken als ePub- en PDF-downloads.

Lees willem.com-posts op je favoriete e-reader (zoals Kobo of reMarkable)

Waarom offline?

Geniet van afleidingsvrij lezen op e-ink-apparaten zoals een Kobo of reMarkable, deze schermen zijn vriendelijk voor je ogen. Concentreer je puur op het artikel, waar en wanneer je maar wilt, want het zijn downloads die volledig offline werken en DRM-vrij zijn. Alle ePubs en PDF's van willem.com bevatten afbeeldingen van hoge kwaliteit en hebben mooie, schaalbare typografie. Ik heb een handig overzicht gemaakt van alle ePub- en PDF-downloads:

Download ePub en PDF

Hoe het werkt

De website willem.com draait op het Lemmid-contentmanagementsysteem, een static site generator die geoptimaliseerde HTML produceert – geschikt voor hoge bezoekersaantallen en internationale distributie. Ik heb de NodeJS-code uitgebreid om automatisch naar ePub en PDF te exporteren, met vereenvoudigde HTML als input. Zie de code-snippets hieronder: voor **ePub** gebruik ik de epub-gen-module; voor **PDF** gebruik ik Pandoc en LaTeX. Ik heb een task manager gebouwd die output-taken in de wachtrij zet voor individuele vertalingen en bestandsformaten, zodat het processingwerk netjes over de backend wordt verdeeld.

Conclusie

De reacties die ik op mijn posts krijg komen uit alle hoeken van de wereld en inspireren me om door te blijven schrijven. Met ePub- en PDF-ondersteuning hoop ik het bereik nog verder uit te breiden door meer manieren te bieden om de artikelen te lezen. Geniet ervan!

```
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 met 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 met Pandoc en LaTeX