Documenten in YAML specificeren en invullen met lua-placeholders in LaTeX

Abstract
Dit artikel bekijkt de implementatie van het factuursjabloon in GinVoice en onderzoekt hoe het factuursjabloon beter kan aansluiten bij het LaTeX-ecosysteem door de introductie van een extra datalaag in YAML met behulp van `lua-placeholders`. Met de introductie van `lua-placeholders` hebben LaTeX-gebruikers volledige vrijheid in het opmaken van factuursjablonen en zijn de factuursjablonen direct integreerbaar met de verbeterde versie van GinVoice.

Keywords
LuaLaTeX, YAML

1 Inleiding

Tijdens mijn werk als Software Engineer werd ik geconfronteerd met een uitdaging voor een bedrijf dat overeenkomsten en voorwaarden opstelt voor meerdere klanten. Een lastige uitdaging daarin was het gescheiden houden van de klantgegevens en regelgevende documentatie. Dit kan, zoals ik eerder had toegepast in GinVoice[3], door extra LaTeX-bestanden te genereren met Python, waardoor vervolgens het LaTeX-hoofdbestand gecompileerd kan worden. Echter, ditmaal was mijn doel om een oplossing te bieden vanuit het LaTeX-domein zelf, in plaats van het applicatiedomein. De oplossing die ik daarvoor heb geschreven heet tegenwoordig `lua-placeholders`[5] en introduceert een gedeelde datalaag met YAML tussen LaTeX en applicatie code. Het pakket biedt een tussenlaag, speciaal bedoeld voor data, door middel van YAML-bestanden. Om deze oplossing te demonstreren pakken we GinVoice. Dit voorbeeld, een Python GTK-applicatie die facturen genereert met LaTeX, biedt net wat meer complexiteit en uitdagingen dan het juridisch domein te bieden heeft.

1.1 De Compiler – LuaLaTeX

Mijn beslissing om LuaLaTeX als compiler te gebruiken, had meerdere redenen. Sinds 2016 maak ik gebruik van LuaLaTeX, wat me destijds voor documenten binnen de computerwetenschap enorm heeft geholpen. In alle afgelopen jaren heb ik veel ervaring opgedaan in het compileren met LuaLaTeX en zie ik het als ontwikkelaar als een geschikte compiler, dankzij de mogelijkheid om in Lua te scripten, wat ik als programmeur uiteraard enorm waardeer.

De mogelijkheid om in Lua te scripten biedt een aantal voordelen. Het stelt mij in staat om complexe taken uit te voeren tijdens het compilatieproces, zoals het verwerken van YAML-bestanden of het manipuleren en structureren van gegevens. Bovendien biedt LuaLaTeX ondersteuning voor Lua init-scripts, waarmee ik een aangepast compilatieproces kan implementeren met een eigen CLI (Command Line Interface), waardoor het integratieproces nog verder wordt vereenvoudigd en geoptimaliseerd voor eindoplossingen.

1.2 Wat is YAML?

Als DevOps Engineer ben ik vaak YAML tegengekomen bij het werken met tools zoals Docker Compose, Travis CI, GitHub Actions en NetPlan van Canonical (Ubuntu systemen). YAML wordt veel gebruikt in de DevOps-wereld voor het automatiseren en beheren van configuraties, waarbij het functioneert als een gestructureerde opmaaktaal voor het definiëren van configuratiebestanden en het vastleggen van infrastructurele en operationele aspecten van softwaretoepassingen.

YAML is een cruciaal onderdeel geworden van moderne softwareontwikkeling en -implementatie, vanwege zijn eenvoudige syntaxis en flexibiliteit. In combinatie met LaTeX biedt YAML een krachtig mechanisme om gestructureerde gegevens te definiëren en te beheren, wat vooral handig is bij het integreren van klantgegevens in LaTeX-documenten. In listing 1 is een voorbeeld van YAML te zien in combinatie met LaTeX.

```
supplier: grapefruit
client: juicing-joker
title: Grapefruit Inc. Invoice
subtitle: for fruits and stuff
currency: \$
number: 1
date: \today
...
```

Listing 1: `invoice-001.yaml`

2 GinVoice

In dit hoofdstuk gaan we kijken naar GinVoice — een open-source Python GTK-applicatie die onder water LaTeX gebruikt om facturen te maken. Daarnaast nemen we de meegeleverde factuurtemplate onder de loep en kijken we naar de betrokken data binnen de factuur.

2.1 De Applicatie

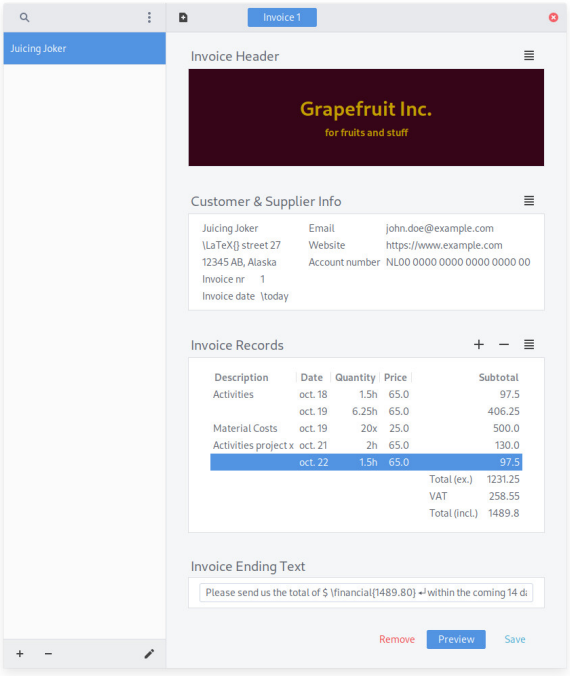
De applicatie heeft meerdere weergaves. De meest voorkomende weergave is de hoofdweergave, waarin je meerdere facturen tegelijkertijd kunt opstellen. In deze weergave, te zien in figuur 1, zijn bijna alle onderdelen terug te zien. Zo zie je de *header*, *informatietabellen*, *factuurregels* en de *slottekst* erin terug komen. In figuur 1 kun je zien dat de invoervelden al zijn ingevuld en dat de inhoud daarvan niet veel afwijkt als je het vergelijkt met het eindresultaat, te zien in figuur 2. Er komen later in dit hoofdstuk nog andere applicatie weergaves naar voren.

2.2 LaTeX-template

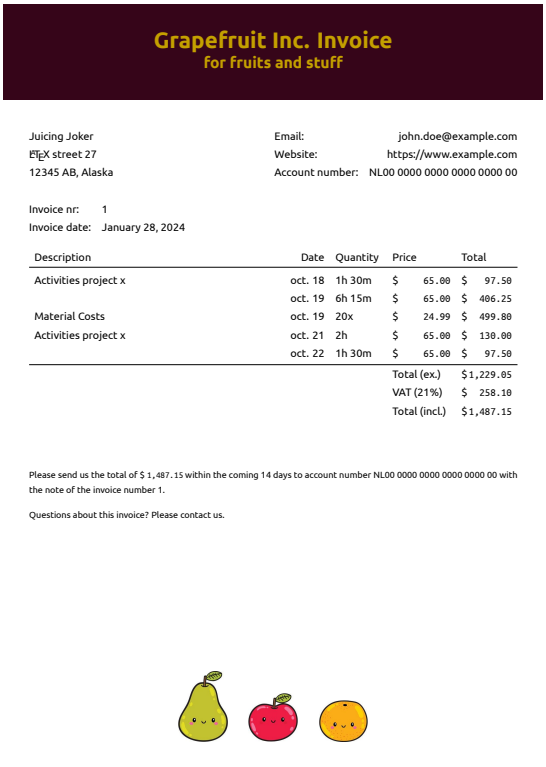
Hieronder vind je een voorbeeld van de code binnen de `document` omgeving:

```
52 \begin{document}
53 \thispagestyle{headermain}
54 \makeheader
55 \vspace{2cm}
56 \begin{tabular}{@{}l@{}}
57    \begin{tabular}{@{}l@{}}
58       \addressee
59    \end{tabular} \\
60    \begin{tabular}{@{}l l@{}}
61       \customerinfo
62    \end{tabular}
63 \end{tabular}
64 \hfill
65 \begin{tabular}{@{}l r@{}}
66    \supplierinfo
67 \end{tabular}\\
68
69 \input{table}
70 \begin{invoice}{\columndef}{\tableheader}{\
       tablefooter}
71    \tablerecords
72 \end{invoice}
73
74 {\footnotesize \theending{}}
75 \vfill
76 \begin{center}
77    \images
78 \end{center}
79
80 \end{document}
```

Listing 2: `invoice.tex`



Figuur 1: GinVoice — de applicatie



Figuur 2: Voorbeeldfactuur gegenereerd met GinVoice

De broncode in listing 2 demonstreert verschillende macro's die vervangen zullen worden door `lua-placeholders`, waaronder `\title`, `\subtitle`, `\addressee`, `\customerinfo`, `\supplierinfo`, `\tablefooter`, `\tablerecords`, `\theending` en `\images`. Daarnaast zijn er variabelen zoals stijl gerelateerde informatie en `\currency` die ook behandeld zullen worden.

2.3 Gegenereerde LaTeX-bestanden

Het is belangrijk om te weten dat GinVoice[3] momenteel een Python script – `generator.py` – gebruikt om extra TeX-bestanden te genereren. Deze TeX-bestanden worden vervolgens in de template ingeladen met `\include`, zodat de benodigde macro's beschikbaar zijn.

Te beginnen met de taalinstelling:

```
\usepackage[english]{babel}
```

Listing 3: `languages.tex`

Destijds heb ik ervoor gekozen om een aparte taalinstelling in de applicatie op te nemen te zien in figuur 3, zodat woorden binnen de factuur netjes worden afgebroken met behulp van `babel`.

Een ander aspect binnen de preamble is het instellen van de documenteigenschappen. Deze macro's komen mee vanuit het gegenereerde bestand `meta.tex`, waarvan de macro's later in het proces worden gebruikt in de `\hypersetup`.

```
\global\def\currency{\$}
\global\def\author{Erik Nijenhuis}
\global\def\title{Grapefruit Inc. Invoice}
\global\def\subject{Invoice for Juicing Joker}
\global\def\keywords{Invoice Grapefruit
    Juicing Joker}
\global\def\producer{GinVoice Generator}
\global\def\creator{gingen}
\global\def\continuationheader{\title{} -- \
    subject{}}
\global\def\continuationfooter{See next page.}
```

Listing 4: `meta.tex`

Veelvoorkomende macro's, zoals `\title` worden op meerdere plekken gebruikt. Dat is dan ook gelijk de reden waarom de `\title` niet in de `header.tex` hoeft te staan.

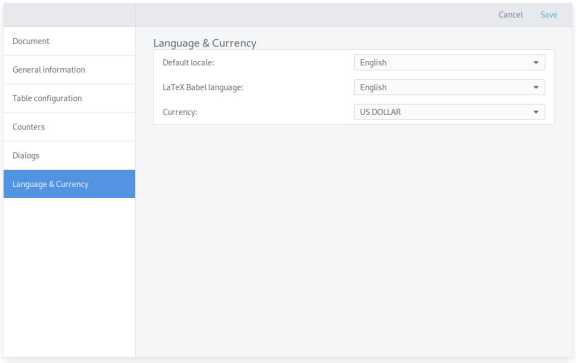
```
\global\def\subtitle{for fruits and stuff}
```

Listing 5: `header.tex`

Het adres van de klant wordt in een macro gestopt, waarbij de adresregels van elkaar zijn gescheiden met behulp van een regeleinde.

```
\newcommand{\addressee}{Juicing Joker\\\LaTeX
    {} street 27\\12345 AB, Alaska}
```

Listing 6: `addressee.tex`



Figuur 3: Taalinstellingen

Deze aanpak zou geschikt zijn voor een tabel met één kolom, of voor bijvoorbeeld een `enumerate` omgeving.

De informatie van de klant en de leverancier gaat uit van een tabel omgeving met twee kolommen.

```
\newcommand{\customerinfo}{
    & \\
   Invoice nr: & 1 \\
   Invoice date: & \today \\
}
```

Listing 7: `customer_info.tex`

```
\newcommand{\supplierinfo}{
   Email: & john.doe@example.com \\
   Website: & https://www.example.com \\
   Account number: & NL00 0000 0000 0000 0000
       00 \\
    & \\
    & \\
    & \\
}
```

Listing 8: `supplier_info.tex`

Het nadeel van deze opzet is dat een en-teken (&) helemaal geen functie heeft binnen de context van de macro zelf. Dat zou pas het geval zijn wanneer je binnen een `tabular` omgeving aan het werk bent. Los van het feit dat de meeste LaTeX-editors hierop een foutmelding geven, werkt deze aanpak gek genoeg alsnog.

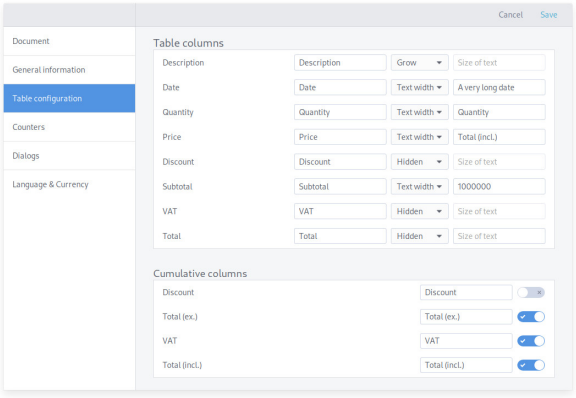
De meest grote uitdaging binnen de applicatie was het configureerbaar maken van de factuurtabel.

Hiervoor is een aparte weergave, te zien in figuur 4. In het figuur zie je dat iedere kolom een andere breedte kan hebben. Dit kan zijn: de lengte van een stuk tekst, zo groot mogelijk, of verborgen. Deze toegevoegde complexiteit vanuit de applicatie heeft destijds

een behoorlijk complexe uitkomst gehad op het gegenereerde `table.tex` bestand, te zien in de volgende code:

```
\newlength{\rowsize}
\setlength{\rowsize}{\linewidth}
\newlength{\cIsize}
\settowidth{\cIsize}{oct. 22}
\addtolength{\rowsize}{-\cIsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newlength{\cIIsize}
\settowidth{\cIIsize}{\textbf{Quantity}}
\addtolength{\rowsize}{-\cIIsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newlength{\cIIIsize}
\settowidth{\cIIIsize}{\textbf{Total (incl.)}}
\addtolength{\rowsize}{-\cIIIsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newlength{\cIVsize}
\settowidth{\cIVsize}{\$ 1,000.00}
\addtolength{\rowsize}{-\cIVsize}
\addtolength{\rowsize}{-2\tabcolsep}
\newcommand{\columncount}{5}
\newcolumntype\columndef{L{1.00\rowsize-2\
    tabcolsep} R{\cIsize} L{\cIIsize} F{\
    cIIIsize} F{\cIVsize}}
\newcommand{\tableheader}{\rowheadercolor{}
    Description&\rowheadercolor{}Date&\
    rowheadercolor{}Quantity&\rowheadercolor{}
    Price&\rowheadercolor{}Total\\}
\newcommand{\tablerecords}{
   Activities project x & oct. 18 & 1h 30m & \
       currency\hfill\financial{65.00} & \
       currency\hfill\financial{97.50}\\
    & oct. 19 & 6h 15m & \currency\hfill\
        financial{65.00} & \currency\hfill\
        financial{406.25}\\
   Material Costs & oct. 19 & 20x & \currency\
       hfill\financial{24.99} & \currency\
       hfill\financial{499.80}\\
   Activities project x & oct. 21 & 2h & \
       currency\hfill\financial{65.00} & \
       currency\hfill\financial{130.00}\\
    & oct. 22 & 1h 30m & \currency\hfill\
        financial{65.00} & \currency\hfill\
        financial{97.50}\\}
\newcommand{\cumoffset}{& & & }
\newcommand{\tablefooter}{\cum{Total (ex.)
    }{1229.05}
\cum{VAT (21\%)}{258.10}
\cum{Total (incl.)}{1487.15}
}
```

Listing 9: `table.tex`



Figuur 4: Tabelinstellingen

Naast de complexe kolom configuratie heb je `\tablerecords` en `\tablefooter`, beide vergelijkbaar met bijvoorbeeld de *leveranciersinformatie*.

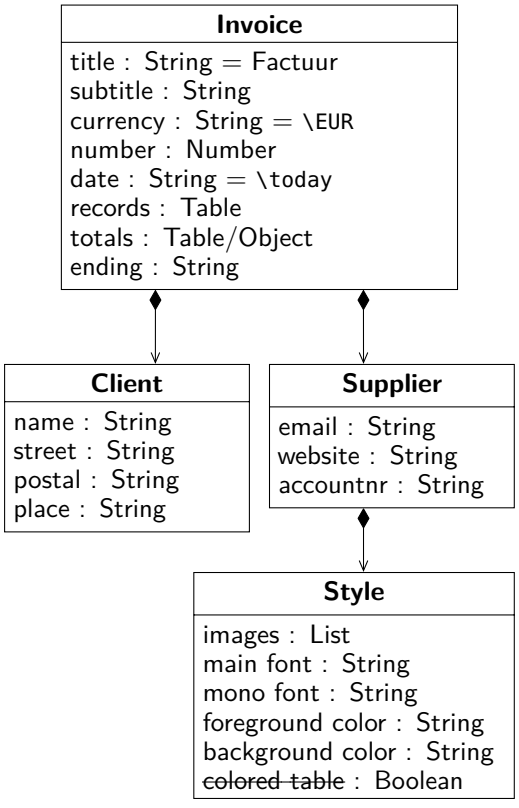
Het laatste gegenereerde bestand `footer.tex` definieert de laatst ontbrekende macro's, `\theending` en `\images`:

```
\newcommand{\theending}{Please send us the
    total of \$ \financial{1487.15}
within the coming 14 days
to account number
NL00 0000 0000 0000 0000 00
with the note of the invoice number 1.

Questions about this invoice?
Please contact us.}
\graphicspath{{/home/erik/.local/share/
    ginvoice/img/}}
\newcommand{\images}{
   \includegraphics[width=.1\textwidth]{image
       1}
   \hspace{1.5em}
   \includegraphics[width=.1\textwidth]{image
       2}
   \hspace{1.5em}
   \includegraphics[width=.1\textwidth]{image
       3}
}
```

Listing 10: `footer.tex`

Destijds heb ik er voor gekozen om alle grafische bestanden ergens binnen de omgeving van GinVoice op te slaan. Dit heb ik weten te koppelen met LaTeX door gebruik te maken van `\graphicspath`.



Figuur 5: Klassendiagram van de factuur

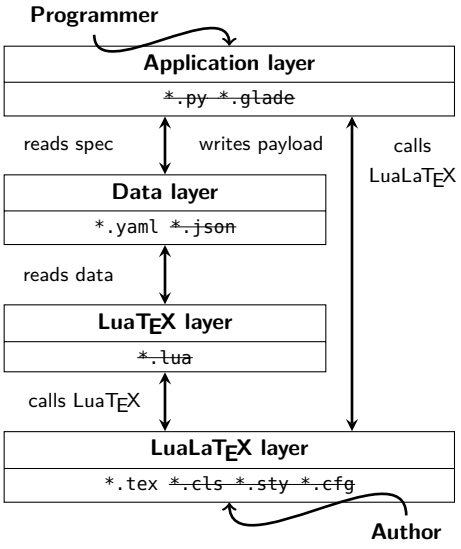
2.4 Factuurdata

Als we kijken naar alle informatie afkomstig uit GinVoice, met uitzonderingen daar gelaten, dan komen we uit op de data gepresenteerd in figuur 5.

Ik heb voor het gemak alvast alle informatie opgeknipt in aparte entiteiten, die overeen zullen komen met de YAML-bestanden, uitvoerig besproken in hoofdstuk 3.

3 Factuursjablonen met `lua-placeholders`

Dit hoofdstuk demonstreert hoe YAML-interfaces, ook wel *recipes* genoemd, kunnen worden gebruikt als interface voor factuursjablonen en hoe dat vervolgens in LaTeX gekoppeld kan worden. Het uiteindelijke doel is om een efficiënte en aanpasbare facturatie-interface te bieden, die gemakkelijk geïntegreerd kan worden in een verbeterde versie van GinVoice. In figuur 6 zie je een representatie van de nieuwe situatie, waarbij alle irrelevante technieken voor dit artikel zijn doorgestreept. De data, te zien in figuur 5, wordt dus van het applicatieniveau naar het dataniveau gebracht. Dit zorgt ervoor dat zowel Python programmeurs als LaTeX-gebruikers kunnen inspelen op het dataniveau, iets wat in de huidige situatie onmogelijk is.



Figuur 6: Niveaus binnen GinVoice

3.1 YAML-specificaties

Aan de hand van de data-analyse in paragraaf 2.4 kunnen we aan de slag met de *recepten*. Alle *recepten* plaatsen we in de map `recipes` relatief vanaf het LaTeX-project. Je zou de map `recipes` ook kunnen bewaren onder `$TEXMFHOME/tex/`, zodat de *recepten* overal beschikbaar zijn.

3.1.1 De Factuur

Het factuur-recept, oftewel `recipes/invoice.yaml`, heeft twee relaties gespecificeerd, namelijk `supplier` en `client`, zoals eerder aangegeven in paragraaf 2.4.

```
1 # Actors
2 supplier:
3   type: string
4 client:
5   type: string
```

Listing 11: `recipes/invoice.yaml`

Hoe de bijbehorende *recepten* worden ingeladen aan de hand van deze waarden wordt beschreven in paragraaf 3.2.3.

De data binnen het factuuronderdeel kan eventueel gestandaardiseerd worden met behulp van een `default` veld, zoals voor `title`. Je kunt zelfs LaTeX aanroepen vanuit een standaardwaarde, zelfs andere parameters met behulp van `\param`.

```
6 # Invoice variables
7 title:
8   type: string
9   default: Factuur \param{number}
10 subtitle:
11   type: string
```

```
  placeholder: Subtitel
currency:
  type: string
  default: \EUR
number:
  type: string
  placeholder: Factuur nummer
date:
  type: string
  placeholder: Factuur datum
```

Naast standaardwaarden kunnen er ook een tijdelijke aanduidingen gespecificeerd worden, ook wel *placeholders* genoemd.

Het meest complexe onderdeel van de factuur is de factuurtabel. Hierin kun je kolommen specificeren, net zoals je gewend bent voor de andere data-types.

```
records:
  type: table
  columns:
    description:
      type: string
    date:
      type: string
    quantity:
      type: string
      default: 0x
    price:
      type: number
      default: 0
    total:
      type: number
      default: 0
```

Zoals eerder vermeld in paragraaf 2.4, kan voor de meeste LaTeX-gebruikers de kolom `total` weggelaten worden en berekend met bijvoorbeeld LaTeX-pakket `invoice2`[2].

Daarnaast is het dan nodig om veld `quantity` van het type `number` te maken en bijvoorbeeld een extra veld `quantity type` toe te voegen, zodat je de juiste notatie kan weergeven voor de `quantity`-kolom.

Voor de eindtotalen heb ik gekozen voor het type `object`, zodat ik de verschillende totalen handmatig in LaTeX kan zetten.

```
totals:
  type: object
  fields:
    total ex:
      type: number
      default: 0
    vat:
      type: number
      default: 0
    total incl:
      type: number
      default: 0
```

De eindtotalen zouden ook op een generiekere manier kunnen, zoals het veld `extra fields` in het leveranciersrecept (zie paragraaf 3.1.3).

Het laatste veld van de factuur `message` maakt gebruik van een speciale YAML-functionaliteit, namelijk *multiline strings* in de standaardwaarde.

```
message:
  type: string
  default: |
    Gelieve het bedrag van
    \currency\ \paramfield{totals}{total
        incl}
    overmaken naar
    \param[supplier]{account number}
    onder vermelding van factuurnummer
    \param{number}
    t.a.v. \param[supplier]{name}.
    \\[2em]

    B.v.d.
```

Met behulp van de *pipe* (`|`) wordt deze modus geactiveerd. Deze constructie is ideaal voor grote teksten, met eventueel LaTeX-syntax.

3.1.2 Klant De klantgegevens bevatten geen bijzondere specificaties ten opzichte van de factuur.

```
name:
  type: string
  placeholder: Klantnaam
street:
  type: string
  placeholder: Straatnaam + nr
postal:
  type: string
  placeholder: 9999 ZZ
place:
  type: string
  placeholder: Plaatsnaam
```

Listing 12: `recipes/client.yaml`

Als alternatief zouden alle adresgegevens kunnen worden gespecificeerd als `list` type, met daarnaast een specificatie, zoals te zien bij `extra fields` bij de leverancier. Dat zou de interface meer generiek maken, echter minder goed aan te passen binnen de LaTeX-context.

3.1.3 Leverancier In het geval van het *recept* voor de leverancier heeft het veld `style` dezelfde functie als `supplier` en `client` van de factuur, waardoor de gebruiker kan kiezen welke stijl wordt toegepast.

```yaml
name:
  type: string
  placeholder: Leveranciersnaam
email:
  type: string
  placeholder: E-mail
website:
  type: string
account number:
  type: string
  placeholder: Rekeningnummer
extra fields:
  type: table
  columns:
    key:
      type: string
    val:
      type: string
# Suppliers style
style:
  type: string
```

Listing 13: `recipes/supplier.yaml`

Een ander interessant veld in deze specificatie is `extra fields`. Dit veld maakt gebruik van het type `table` om extra informatievelden toe te staan, zoals bijvoorbeeld het rekeningnummer van de leverancier, het BTW-nummer, of andere relevante details. Het gebruik van een tabel in plaats van een vast aantal velden geeft de eindgebruiker de flexibiliteit om zoveel extra informatie toe te voegen als nodig is, zonder beperkingen op te leggen.

3.1.4 Stijl In het stijl-recept kunnen fonts, kleuren en meerdere plaatjes gespecificeerd worden. Zoals eerder aangegeven: voor LaTeX-gebruikers zou dit volledig gespecificeerd kunnen worden in LaTeX zelf. Het stijl-recept kan dan worden weggelaten.

```yaml
images:
  type: list
  item type: string
main font:
  type: string
  default: Ubuntu
mono font:
  type: string
  default: Ubuntu Mono
foreground color:
  type: string
  default: 000000
background color:
  type: string
  default: FFFFFF
```

Listing 14: `recipes/style.yaml`

Opmerkelijk is het type voor `images`, namelijk `list`. In paragraaf 3.3 is te zien hoe deze lijst onderaan de factuur wordt ingeladen.

3.2 De Nieuwe Factuur

Nu de *recepten* op orde zijn kunnen we over stappen naar het integreren binnen LaTeX.

3.2.1 Recepten Inladen in de Preamble De *recepten* worden ingeladen met behulp van de macro `\loadrecipe`.

```
\loadrecipe[\jobname]{recipes/invoice.yaml}
\loadrecipe{recipes/supplier.yaml}
\loadrecipe{recipes/client.yaml}
\loadrecipe{recipes/style.yaml}
```

Je ziet voor het `invoice`-recept dat het de ⟨*namespace*⟩ `\jobname` krijgt. Dit heeft te maken met het feit dat macro `\param` standaard `\jobname` als ⟨*namespace*⟩ gebruikt, wat het gebruik versimpelt.

De andere *recepten* geven geen ⟨*namespace*⟩ op, wat betekent dat ze de 'basename' van het pad als ⟨*namespace*⟩ dragen. In dit geval respectievelijk `supplier`, `client` en `style`.

3.2.2 Munteenheid Voor wat betreft de munteenheid heb ik ervoor gekozen om die in de macro `\currency` te vermommen. Dit is vanwege het feit omdat het ook in andere bestanden gebruikt wordt, zoals `invoice.cls`.

```
\def\currency{\rawparam{\jobname}{currency}}
```

Mocht de ⟨*currency*⟩ niet gezet zijn, dan wordt de standaardwaarde van `style.yaml` gebruikt. In dit geval komt het dan neer op `\EUR`.

3.2.3 Waarden Inladen Alle YAML-bestanden met betrekking tot de *payload* administreer ik in corresponderende mappen.

```
⟨projectnaam⟩
├── recipes
│   └── ⟨recept⟩.yaml
├── invoices
│   └── ⟨invoice-xxx⟩.yaml
├── clients
└── et cetera
```

Waarden, ook wel de *payload* genoemd, worden net als recepten ingeladen, maar dan met de macro `\loadpayload`. Dankzij de relaties, beschreven in paragraaf 2.4, is het net wat complexer dan *recepten*, omdat `lua-placeholders` hier niet iets standaard voor aanbiedt.

```
\IfFileExists{invoices/\jobname.yaml}{
    \loadpayload[\jobname]{invoices/\jobname
        .yaml}
    \strictparams
}{}
```

Voor het inladen van de factuurwaarden wordt namelijk gecontroleerd of er een corresponderend YAML-bestand voor bestaat. Mocht dat het geval zijn, dan wordt die *payload* ingeladen, en wordt er gebruik gemaakt van de experimentele macro `\strictparams`, wat in de toekomst zal inhouden dat er errors optreden bij het ontbreken van verplichte data.

In geval er geen corresponderend bestand voor is gevonden, wordt er een factuursjabloon gecompileerd.

Na het inladen van de factuurdata kunnen we kijken of er een klant is gespecificeerd in de factuurdata. Dit doen we met behulp van `\hasparam`. Het gaat om de factuurdata, waarvoor we geen ⟨*namespace*⟩ hoeven op te geven.

```
\hasparam{client}{%
    \loadpayload[client]{clients/\rawparam{\
        jobname}{client}.yaml}
}{}
```

Over het algemeen is `\param` niet bedoeld voor gebruik binnen de preamble, omdat het ook placeholders met LaTeX-opmaak kan opleveren. Voor dit soort lastige situaties is de macro `\rawparam` geschreven, zoals gedaan voor de klant en leverancier. Deze macro heeft geen optionele argumenten, wat vaak problemen oplevert met bijvoorbeeld `pgfkeys`.

```
\hasparam{supplier}{%
    \loadpayload[supplier]{suppliers/\
        rawparam{\jobname}{supplier}.yaml}
}{}
```

Zoals te zien is, verschilt het inladen van de leverancier niet van het inladen van de klant. Echter, komt er nog wel een vervolgactie na het inladen van de leverancier, namelijk kijken of de stijl kan ingeladen worden. Dit gaat in de basis hetzelfde als bij de klant en leverancier zelf, echter zie je hier dat de ⟨*namespace*⟩ wel gezet moet worden.

```
\hasparam[supplier]{style}{%
    \loadpayload[style]{styles/\rawparam{
        supplier}{style}.yaml}
    \setmainfont{\rawparam{style}{main font
        }}
    \setmonofont{\rawparam{style}{mono font
        }}
    \definecolor{backgroundcolor}{HTML}{\
        rawparam{style}{background color}}
    \colorlet{bgcolor}{backgroundcolor}
    \definecolor{foregroundcolor}{HTML}{\
        rawparam{style}{foreground color}}
    \colorlet{textcolor}{foregroundcolor}
}{}
```

Voor de stijlgerelateerde data heb ik gekozen om de waarden gelijk te configureren in de bijbehorende macro's, zoals `\setmainfont` en `\definecolor`, zolang er maar een stijl is opgegeven. Je kunt er ook voor kiezen om standaard alsnog de stijlwaarden te zetten aan de hand van de standaardwaarden opgegeven in het `style`-recept, door de configuratie daarvan buiten het `\hasparam`-blok te zetten.

3.3 Verwerken in Document

Voordat we over kunnen gaan naar het compileren van facturen rest ons nog één taak, namelijk alle waarden in het document zelf zetten.

3.3.1 Header Zoals eerder aangegeven in hoofdstuk 2 is de macro `\makeheader` afkomstig van `invoice.cls`. Voor nu gaan we er van uit dat `invoice.cls` zodanig is aangepast dat die geen fouten meer veroorzaakt en dat `\makeheader` nu de titel en subtitel als argumenten verwacht:

```
\begin{document}
\thispagestyle{headermain}
\makeheader{\param{title}}{\param{subtitle
    }}
\vspace{2cm}
```

In het voorbeeld zijn er maar twee verschillen met betrekking tot de nieuwe versie, namelijk:

`\title` → `\param{title}`
`\subtitle` → `\param{subtitle}`

Echter, dit keer op een juiste manier doorgegeven aan de `documentclass`.

3.3.2 Informatie De linker kolom van de informatie is vrij lastig, aangezien het klantinformatie bevat, maar ook factuurgegevens, zoals het nummer en de datum.

```
\begin{tabular}{@{}l@{}}
    \begin{tabular}{@{}l@{}}
       \param[client]{name}\\
       \param[client]{street}\\
       \param[client]{postal}, \param[client
           ]{place}\\
    \end{tabular} \\
    \begin{tabular}{@{}l l@{}}
       Factuur nummer: & \param{number}\\
       Factuur datum: & \param{date}\\
    \end{tabular}
\end{tabular}
\hfill
```

Je ziet bij de adresregels dat er per regel een regeleinde wordt gezet. Dit had ook gekund als er bijvoorbeeld een veld `address lines` van het type `list` was. Dan was dat met `\param[client]{address lines}` in één keer opgelost, gegeven dat `postal` en `place` zijn samengevoegd op één adresregel in YAML. Het genoemde alternatief gaat er wel vanuit dat de macro `\paramlistconjunction` is gezet naar '\\', in plaats van de standaardwaarde ', ~'.

```
92 \begin{tabular}{@{}l r@{}}
93    Bedrijf: & \param[supplier]{name} \\
94    E-mail: & \param[supplier]{email} \\
95    Website: & \param[supplier]{website} \\
96    Rekening-nr: & \param[supplier]{account
          number} \\
97    \hasparam[suplier]{extra fields}{%
98       \def\formatsupplierextra{\key & \val
             \\}%
99       \fortablerow[supplier]{extra fields
             }{formatsupplierextra}
100   }{}
101 \end{tabular}\\
```

De rechter kolom van informatie is vergelijkbaar met de linker, alleen heeft het nog een bijzonder veld, namelijk `extra fields` van het type `table`. Op deze manier kan er een variabel aantal rijen toegevoegd worden. Hetzelfde zou eventueel toegepast kunnen worden voor de klantgegevens in de linker kolom. Dan rest alleen nog de keuze of je ze boven of onder de factuurinformatie plaatst.

3.3.3 *Tabel*

Zoals eerder genoemd is de kolomdefinitie lastig te standaardiseren. Op regel 105 is te zien wat de `\columdefs` had kunnen geven, met uitzondering op de counters die ik daarvoor eerder inzette.

```
103 \begin{invoice}
104 % Column Definition based on 540pt
105 {@{}L{180pt-\tabcolsep} R{80pt-2\tabcolsep
        } L{60pt-2\tabcolsep} F{120pt-2\
        tabcolsep} F{100pt-\tabcolsep}@{}}
```

Voor het tweede argument van de `invoice`-omgeving is een statische header gezet.

```
106 % Header
107 {\textbf{Beschrijving} & \textbf{Datum} &
        \textbf{Aantal} & \textbf{Prijs} & \
        textbf{Totaal} \\ \hline}
```

Voor het derde argument van de `invoice`-omgeving is te zien hoe de eindtotalen gezet worden in de tabel. Deze totalen worden iedere keer in de laatste twee kolommen van een rij geplaatst, zodat het netjes uitlijnt met de rest van de tabel.

```
108 % Totals
109 {%
110    & & & \textbf{Totaal (ex.)} & \currency
          \hfill{\ttfamily\paramfield{totals
          }{total ex}} \\
111    & & & \textbf{BTW} & \currency\hfill{\
          ttfamily\paramfield{totals}{vat}}
          \\
112    & & & \textbf{Totaal (incl.)} & \
          currency\hfill{\ttfamily\paramfield
          {totals}{total incl}} \\
113 }
```

In het laatste onderdeel van de tabel is te zien hoe aan de hand van `\formatrecords` iedere factuurregel wordt gezet met behulp van `\fortablerow`.

```
114    \newcommand\formatrecords{%
115       \description & \date & \quantity &%
116       \currency\hfill{\ttfamily\price} &%
117       \currency\hfill{\ttfamily\total} \\}
118    \fortablerow{records}{formatrecords}
119 \end{invoice}
```

De algehele structuur van de tabel is nog afkomstig uit de vorige situatie. Het opmerkelijke verschil ten opzichte van de oude situatie is dat de data in allerlei tabelstructuren gezet kan worden, aangezien de data losgekoppeld is van het LaTeX- en applicatiedomein en de uitdagingen op het gebied van typesetting zijn verlegd naar het LaTeX-domein.

3.3.4 *Slottekst en Plaatjes*

Waar we eerder een geavanceerde YAML-specificatie zagen voor het veld `message`, blijft de implementatie binnen LaTeX vrijwel hetzelfde:

```
121 {\footnotesize\param{message}}
```

Het enige verschil is:

`\theending` → `\param{message}`

De plaatjes, daarentegen, zijn iets lastiger te implementeren in LaTeX dankzij het type `list`.

```
122 \newcommand\formatimage[1]{\hspace{.75em}\
        includegraphics[width=2cm]{#1}\hspace
        {.75em}}%
123 \hasparam[style]{images}{%
124    \vfill
125    \begin{center}
126       \forlistitem[style]{images}{
             formatimage}
127    \end{center}
128 }{}
129 \end{document}
```

Waar eerder in Python alle plaatjes netjes naast elkaar gezet werden, waarbij tussen elk plaatje een `\hspace` van `1.5em`, heb ik ervoor gekozen om aan beide kanten van ieder plaatje de helft daarvan als `\hspace` op te voeren. Dit heeft te maken met de macro `\forlistitem`, die daar nog geen handige manier voor heeft, zoals `\param` dat wel heeft door het zetten van `\paramlistconjunction` naar '`\hspace{1.5em}`'.

4 Uitvoering

Nu de legacy factuur volledig is getransformeerd, kunnen we gaan kijken wat het resultaat daarvan is. Mocht je mee willen doen via de commandline, bekijk dan de volledige source-code[4] van deze voorbeelden.

4.1 De Template Versie

Zonder enige waarden aan te leveren krijgen we het volgende resultaat, te zien in figuur 7.

Zoals eerder gezegd is `lua-placeholders` alleen met LuaLaTeX te compileren. Het voorbeeld kan als volgt gecompileerd worden:

```
lualatex --jobname=invoice-template \
    --output-directory=$(OUTPUT_DIR) \
     invoice
```

Listing 15: Compileren met `lualatex`

Waar `$(OUTPUT_DIR)` de gewenste uitvoermap is.

Echter, als je een template aan het opmaken bent, dan is blijvend genereren met `latexmk`[1] gebruiksvriendelijker:

```
latexmk -pvc -lualatex \
   --jobname=invoice-template \
   --output-directory=$(OUTPUT_DIR) \
    invoice
```

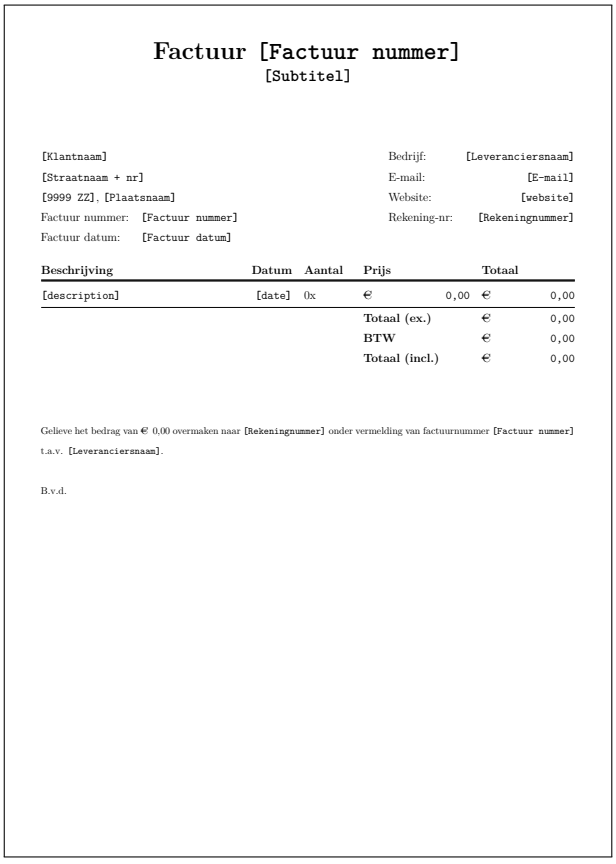
Listing 16: Compileren met `latexmk`

Zo hoef je bij iedere wijziging in TeX niet opnieuw te compileren, maar gebeurt dat vanzelf.

4.2 YAML-waarden

Om een ingevulde factuur te krijgen zullen we de volgende YAML-bestanden nodig hebben:

```
⟨project dir⟩
├── invoices
│   └── ⟨invoice⟩.yaml
├── suppliers
│   └── ⟨supplier⟩.yaml
├── styles
│   └── ⟨style⟩.yaml
└── clients
    └── ⟨client⟩.yaml
```



Factuur [Factuur nummer]

[Subtitel]

[Klantnaam]
[Straatnaam + nr]
[9999 ZZ], [Plaatsnaam]
Factuur nummer: [Factuur nummer]
Factuur datum: [Factuur datum]

Bedrijf: [Leveranciersnaam]
E-mail: [E-mail]
Website: [website]
Rekening-nr: [Rekeningnummer]

Beschrijving	Datum	Aantal	Prijs	Totaal
[description]	[date]	0x	€ 0,00	€ 0,00
			Totaal (ex.)	€ 0,00
			BTW	€ 0,00
			Totaal (incl.)	€ 0,00

Gelieve het bedrag van € 0,00 overmaken naar [Rekeningnummer] onder vermelding van factuurnummer [Factuur nummer] t.a.v. [Leveranciersnaam].

B.v.d.

Figuur 7: `invoice-template.pdf`

Deze structuur is gebaseerd op de implementatie beschreven in paragraaf 3.2.3. Voordat de inhoud van de YAML-bestanden besproken wordt, kunnen we eerst alternatieve projectstructuren overwegen.

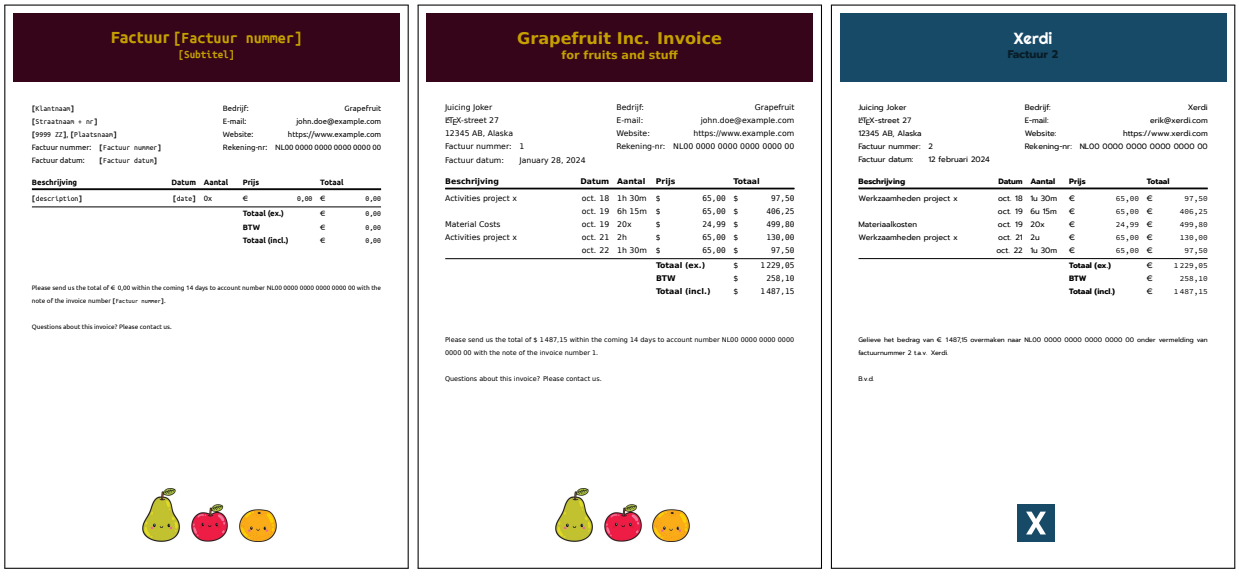
4.2.1 Alternatieve Projectstructuur

Het staat iedereen natuurlijk vrij om zelf een gewenste mappenstructuur aan te leggen. Zo zou je bijvoorbeeld styles onder

`/suppliers/`⟨*supplier*⟩`/style.yaml`

kunnen zetten, zodat je zelfs het `style` veld kunt weglaten in het leveranciers-recept. Een andere optie is de `clients` map onder leveranciersniveau te plaatsen, zodat je niet per ongeluk klanten van leveranciers door elkaar haalt. Dit zou je kunnen realiseren door het volgende te doen:

```
⟨project dir⟩
└── suppliers
    ├── ⟨supplier⟩.yaml
    └── ⟨supplier⟩
        └── ⟨client⟩.yaml
```



(a) `invoice-template.pdf` (b) `invoice-001.pdf` (c) `invoice-002.pdf`

Figuur 8: Factuurvoorbeelden

Op deze manier zou de implementatie voor het inladen van klanten de variabelen ⟨*supplier*⟩ en ⟨*client*⟩ vereisen, om tot het volgende pad te komen:

`suppliers/`⟨*supplier*⟩`/`⟨*client*⟩`.yaml`

Dezelfde overweging zou voor de facturen toegepast kunnen worden, echter is dit een lastiger scenario, aangezien de factuurdata wordt gebaseerd op de `\jobname` in de implementatie van paragraaf 3.2.3. Een mogelijke oplossing hiervoor is om het project per leverancier te beheren. Je kunt dan de *recepten* in de map `$TEXMFHOME/tex` zetten, zodat deze voor alle projecten aanwezig zijn. Hier een voorbeeld van een mogelijke projectstructuur:

```
$HOME/texmf/tex
├── recipes
│   ├── invoice.yaml
│   ├── client.yaml
│   ├── supplier.yaml
│   └── style.yaml
├── invoice.cls
└── invoice.tex
```

```
⟨project dir⟩
├── invoices
│   └── ⟨invoice⟩.yaml
├── clients
│   └── ⟨client⟩.yaml
├── supplier.yaml
└── style.yaml
```

In dit voorbeeld is alle data per leverancier gescheiden, waaronder de klantinformatie en de uiteindelijke facturen.

4.3 Leveranciers en Klanten

In het voorbeeld resultaat van GinVoice was een klant *Juicing Joker* te zien. In YAML zou dat neer komen op:

```
name: Juicing Joker
street: \LaTeX-street 27
postal: 12345 AB
place: Alaska
```

Listing 17: `clients/juicing-joker.yaml`

Op deze manier kan de klant aangehaald worden in de factuur met `juicing-joker`.

Voor de leverancier zagen we *Grapefruit Inc.* in het voorbeeld, wat neerkomt op:

```
name: Grapefruit
email: john.doe@example.com
website: https://www.example.com
account number: NL00 0000 0000 0000 0000 00
style: grapefruit
```

Listing 18: `suppliers/grapefruit.yaml` of `grapefruit/supplier.yaml`

En voor de stijl:

```
main font: DejaVu Sans
mono font: DejaVu Sans Mono
foreground color: c4a000
background color: 360519
images:
  - img/image1
  - img/image2
  - img/image3
```

Listing 19: `styles/grapefruit.yaml` of `grapefruit/style.yaml`

Het voordeel van de alternatieve projectstructuur is dat `invoice-template` automatisch de styling oppakt evenals de leveranciersinformatie, te zien in figuur 8a.

4.4 Facturen

Om een factuur te maken die exact overeenkomt met het standaard voorbeeld van GinVoice, zoals te zien in figuur 8b, maken we gebruik van het volgende YAML-voorbeeld:

```yaml
supplier: grapefruit
client: juicing-joker
title: Grapefruit Inc. Invoice
subtitle: for fruits and stuff
currency: \$
number: 1
date: January 28, 2024
records:
  - description: Activities project x
    date: oct. 18
    quantity: 1h 30m
    price: 65
    total: 97.5
  - description: ''
    date: oct. 19
```

Listing 20: `invoices/invoice-001.yaml`

De actoren `grapefruit` en `juicing-joker`, besproken in paragraaf 4.3, zien we terug in de factuur. Daarnaast heeft het voorbeeld dezelfde algemene informatie om tot hetzelfde resultaat te komen. Bij het veld `records` zie je dat een rij van de tabel veel regels inneemt. Bij de tweede rij van de tabel zie je dat het veld `description` een lege waarde heeft.

Mochten de quotes weggelaten worden in YAML, dan krijg je een fout bij het converteren naar data. Aangezien de rijen niet al te veel verschillen van elkaar, vervolgen we het voorbeeld bij het veld `totals`:

```yaml
totals:
  total ex: 1229.05
  vat: 258.10
  total incl: 1487.15
message: |
  Please send us the total of \currency~\
      paramfield{totals}{total incl}
  within the coming 14 days to account
      number
  \param[supplier]{account number} with the
       note of the invoice number \param{
      number}.\\[2em]

  Questions about this invoice? Please
      contact us.
```

Als laatste in het voorbeeld zien we de totalen en de slottekst. Deze factuur kan vervolgens gecompileerd worden met de volgende opdracht:

```
lualatex --jobname=invoice-001 \
   --output-directory=$(OUTPUT_DIR) \
    invoice
```

5 Conclusie

In dit artikel hebben we niet alleen de implementatie van factuursjablonen in GinVoice onderzocht, maar ook een innovatieve methode voorgesteld om deze sjablonen naadloos te integreren met het LaTeX-ecosysteem. Door YAML te gebruiken als tussenlaag en `lua-placeholders` voor dynamische invoegingen, hebben we niet alleen een robuuste en flexibele oplossing geboden voor het genereren van facturen, maar ook een raamwerk gecreëerd waarin verschillende documentonderdelen, zoals klantinformatie, document-overkoepelend kunnen worden ingezet.

Deze aanpak biedt LaTeX-gebruikers niet alleen de vrijheid om factuursjablonen naar wens aan te passen, maar opent ook de deur naar een breder scala aan toepassingen. Door dezelfde YAML-gebaseerde structuur te gebruiken, kunnen verschillende documenten, zoals contracten en facturen, met gemak worden gegenereerd en onderhouden. Dit vergroot niet alleen de consistentie tussen verschillende documenttypen, maar verhoogt ook de efficiëntie van het documentatieproces als geheel.

Het gebruik van `lua-placeholders` in combinatie met YAML maakt het mogelijk om dynamische content toe te voegen aan sjablonen, waardoor gebruikers een meer gestroomlijnde workflow ervaren. Deze flexibiliteit maakt het gemakkelijk om gegevens en opmaak van verschillende documenten te scheiden, terwijl het tegelijkertijd de mogelijkheid biedt om deze onderdelen document-overkoepelend in te zetten.

Als conclusie biedt deze aanpak niet alleen een waardevolle bijdrage aan het optimaliseren van factureringsprocessen, maar opent het ook nieuwe mogelijkheden voor het efficiënt genereren en beheren van verschillende soorten documenten binnen een organisatie.

6 Discussie

6.1 LaTeX Compilers

In het artikel ga ik uit van de LuaLaTeX-compiler. Voor LaTeX-gebruikers die andere compilers gebruiken, biedt `lua-placeholders` geen oplossing. Hoewel sommige compilers alsnog ondersteuning bieden voor Lua, houdt `lua-placeholders` hier geen rekening mee. Onderzoek en implementatie zou het draagvlak van `lua-placeholders` voor de LaTeX-gemeenschap kunnen verbeteren.

6.2 JSON vs YAML

Op de keuze van YAML in plaats van JSON ben ik niet ingegaan in het artikel. Beide zijn bedoeld voor data en hoewel JSON bekender is en meer compatibiliteit heeft met programmeertalen, heb ik juist voor YAML gekozen in belang van de leesbaarheid van LaTeX-broncode. Zoals uitvoerig gedemonstreerd, bevatten de bestanden

veel LaTeX-broncode. Bij gebruik van JSON zou iedere backslash ge-*escaped* moet worden.

Vergelijk bijvoorbeeld:

```
title: Factuur \param{number}
```

en

```
{ "title": "\\param{number}" }
```

Voor een LaTeX-gebruiker lijkt het mij prettiger om waarden aan te passen in YAML voor testdoeleinden, dan in JSON.

6.3 Roadmap GinVoice

De ontwikkeling heeft enige tijd stil gestaan, echter ben ik kort geleden erachter gekomen dat de oplossing ook voor Windows platformen kan werken. Het brengen van GinVoice naar het Windows platform verbreedt de doelgroep aanzienlijk en zou naar mijn verwachting meer draagvlak voor LaTeX kunnen opleveren.

Voor wat betreft het introduceren van `lua-placeholders` zijn er nog een paar obstakels te overbruggen, zoals uitdagingen met betrekking tot vertaling en de niet besproken variabele kolomdefinitie, wat nu juist een gebruiksvriendelijk onderdeel van de applicatie is.

Referenties

[1] J. Collins, E. McLean, D.J. Musliner. The latexmk package. `https://www.cantab.net/users/johncollins/latexmk/index.html`

[2] S. Dierl. The invoice2 package. `https://github.com/no-preserve-root/invoice2`

[3] E. Nijenhuis. The GinVoice GTK application. `https://gitlab.gnome.org/MacLotsen/ginvoice`

[4] E. Nijenhuis. The GinVoice Template. `https://github.com/Xerdi/ginvoice-template/tree/maps`

[5] E. Nijenhuis. The lua-placeholders package. `https://ctan.org/pkg/lua-placeholders`

Erik Nijenhuis
erik (at) xerdi dot com
`https://github.com/MacLotsen`