LaTeX-on-HTTP: LaTeX as a commodity web service for application developers

Yoan Tournade

Abstract

Although LaTeX is widely used in academia and education, only a few developers use it to create PDFs in web applications or IT systems. This seems strange considering that many developers are well exposed to these academic and scientific milieux — and that many recognize LaTeX for its superior typesetting qualities.

In this paper, we try to answer two questions:

- Why use of LaTeX in IT systems is not widespread — and not even common?
- How can we change this state of affairs?

In this article we give an overview of LaTeX-on-HTTP (`github.com/YtoTech/latex-on-http`), an attempt to commodify and simplify LaTeX use in web applications or IT systems, and to ease its adoption by modern developers.

1 Introduction

LaTeX has its indisputable niches, like academic and scientific publications. Thanks to web projects like Overleaf (`overleaf.com`), it has also become more readily accessible for the lay computer user to complete common editing chores.

I found it strange then that one of the populations most exposed to LaTeX, and also the most likely to recognize its merits, *the computer engineers*, do *not* use LaTeX in the applications they develop — and that they may not even think to do so.

Currently a modern application developer that needs PDF output for a project will find an abundance of solutions by searching the web:

1. HTML/CSS to PDF converters, e.g., wkhtmltopdf (`wkhtmltopdf.org`) and WeasyPrint (`weasyprint.readthedocs.io`);
2. instruction-based PDF generators, e.g., PDFKit (`pdfkit.org`) in Node, and FPDF (`fpdf.org`) in PHP;
3. headless calls to browsers or office software, e.g., Puppeteer (`github.com/puppeteer`) is an example of a headless API to run Chrome/Chromium browsers with no GUI.

Browsing a couple of *Stack Overflow* entries will give our hypothetical developer copy-paste ready initial working code, in his language of interest. LaTeX will not appear in the results; and may well not pop up in our developer's mind as an eligible solution.

All these tools provide decent results, but we argue they are far from the typesetting quality attainable by LaTeX. While we can understand the prevalence of the first class of solutions above by the opportunity to leverage existing HTML and CSS code, this does not explain why LaTeX would be virtually absent from the common set of solutions.

2 Why LaTeX is uncommon in modern applications for PDF creation

We explain this state of affairs from three viewpoints: techniques, customs and knowledge.

2.1 Technical barriers: the not-so-accessible LaTeX runtime

It is easy enough to install a LaTeX distribution on a personal computer. Modern browser-based services like CoCalc (`cocalc.com`) [4] and Overleaf have even removed this need and drastically reduced the time required to first interact with LaTeX [2, 3]. Tools such as MathJax (`mathjax.org`) and LaTeX Base (`latexbase.com`) [1] have even demonstrated that LaTeX (or a subset) can be run in a browser.

However the requirements of an application developer are different: he does indeed want convenience, but even more importantly he needs reproducibility and scaling — and he certainly does *not* want to battle with his infrastructure team to explain why it would be pertinent to add many hundred of megabytes of complex runtime requirements to generate PDFs in their applications. Many developers do not even fully control their runtime environment (restricted cloud, etc.).

When the developer has only a few hours, or at best days, to automate PDF creation, he will shy away from the complexities of adding the LaTeX runtime in his application — he has to go for the safer and more recognized solutions.

2.2 Cultural glass walls: mental buckets of use cases

Developers, often coming or having been exposed to academic and scientific milieux, know of LaTeX and recognize its typesetting superiority. They are the people who might tease one of their mates for *not* using LaTeX in their latest technical publication.

However, for them LaTeX is in a mental bucket that excludes their web application development or IT system integration activities. More often than not, they will not think about LaTeX for making a PDF document in their application: a glass wall of custom separates their need from the LaTeX solution.

2.3 Imperfect knowledge bases: learning by copying

Even if our developer considers LaTeX as a prospective solution, he must then learn about its implementation.

As developers, we try not to reinvent the wheel with each task. We search and we find shared knowledge about recommended tool usage and common patterns. Generally, we use readily available and copy-pasteable code samples for our needs; and tweak our custom solution from there.

Our developer could easily find a proper LaTeX template to start from for his PDF application — there are great resources on the web for that. But then? Our developer will need the code to compile their LaTeX template to a PDF, which they will not easily find on the web. Compile a LaTeX document on my computer or on the web? Easy. Compile a LaTeX template from a web application — either from a server or from any browser[1] — in a couple of lines of code and without adding potentially problematic requirements? Not so easy.

Few pertinent examples are available to lead our developer to a way to implement LaTeX in his application. If he has the time to be curious and is up for a challenge, he can find a path [5, 6]. This is especially so in the niches where the need for very high-quality or specialized document creation will justify the cost of bringing LaTeX into the infrastructure. But this is not the typical case that interests us here: we consider the common modern web developer that at first just wants to create a rather simple PDF document in his application.

3 Introducing LaTeX-on-HTTP

LaTeX-on-HTTP first emerged when I needed a way to compile LaTeX documents to automate invoicing, without installing the whole LaTeX stack.[2]

The basic requirements for LaTeX-on-HTTP were rather straightforward: take the TeX Live runtime, put it on a server, and add an HTTP-based API between the user and the server so we can pass the files to be compiled. *Voilà* — now users just need Internet access to generate their PDF documents; they do not even need to know that they are using LaTeX.

[1] And considering the disparate array of web browsers, this is not the least of requirements.

[2] After converting my quotes-and-invoices consulting templates to LaTeX, I was glad of the result, but I wanted my other team members to be able to make these PDF documents without them having to install gigabytes of additional software, or me having to support them for managing missing CTAN packages.

3.1 HTTP: the web *lingua franca*

The HTTP API is an important part of the solution; it gives us several desirable properties:

- as HTTP is ubiquitous in modern applications, the service is accessible from most of the technical stacks: no more need for complex runtime requirements, a simple HTTP client suffices;
- the HTTP protocol is well-known to most developers;
- it clearly advertises the solution as intended for automation and integration.

LaTeX-on-HTTP is not the first solution to put the LaTeX stack behind an HTTP API,[3] but it may be the first designed with developers as first-class users.

3.2 Hello world: specifying a compilation job with JSON

Let's now present how to use the LaTeX-on-HTTP for compiling documents in applications.

The following JSON[4] code is sent in a POST HTTP request to the `/builds/sync` endpoint:[5]

```
{
"compiler": "lualatex",
"resources": [
    {
        "main": true,
        "content": "\\documentclass{article}
        \\usepackage{graphicx}
        \\begin{document}
        Hello World
        \\includegraphics[width=5cm]{logo.png}
        \\end{document}"
    },
    {
        "path": "logo.png",
        "url": "https://www.ytotech.com/static/
images/ytotech_logo.png"
    }
]
}
```

(The line break in the `url` value is editorial.)

This request will return a PDF file if the compilation succeeds. If there is an error, the API will return a JSON payload including the compiler logs.

As we can see, we pass two main things to the LaTeX-on-HTTP endpoint:

- a set of resources — or source files — to be compiled, with the path specified for each;
- the engine selected — and other potential options — to control the compilation environment.

We may also remark that we can use different means to transfer the resources to be compiled: here, by passing the string content directly, or by providing a url pointing to a file. We can imagine and provide several other ways, for convenience and for supporting various use cases.

It is essential to normalize as much as possible the LaTeX compilation job input for ensuring the reproducibility of the process and to provide further capabilities, such as caching of input resources or output documents.

3.3 Real-life example: editing a letter

A developer can use this API to construct a real-life application. Let's say our developer wants to edit a personal letter from a bank IT system, notifying customers of their account opening. Using Python as our language, the code could look like the following:

```
# Open and read the template LaTeX file.
with open("opening-letter-template.tex") as f:
    template_str = f.read()

# Replace the dynamic content.
template_str = template_str.replace(
    "<<letter-title>>, "Your account...")
template_str = template_str.replace(
    "<<letter-body>>, "Dear Mr...")

# Loads a binary image file.
with open("duck_logo.png", "rb") as f:
    logo_binary = f.read()
    # Convert to base64.
    logo_b64 = base64.b64encode(
        logo_binary).decode("utf-8")

# Generate the PDF file,
# using an HTTP client (requests).
r = requests.post(
    "https://latex.ytotech.com/builds/sync",
    json={
        "compiler": "pdflatex",
        "resources: [
            {
                "main": true,
                "content": template_str
            },
            {
                "path": "logo.png",
                "file": logo_b64,
            },
```

[3] ShareLaTeX/Overleaf's CLSI (`github.com/overleaf/clsi`) and Andrey Lushnikov's LaTeX-Online (`github.com/aslushnikov/latex-online`) are significant open source precedents.

[4] JSON (`json.org`) has become a dominant force in data serialization languages in web applications and APIs, rivaling or surpassing XML.

[5] You can easily try a similar example on the command line of your computer by using `curl`. A snippet is available on the project page: `github.com/YtoTech/latex-on-http`.

```
        ]
    }
)

# Save the PDF output.
with open("opening-letter.pdf", "wb") as f:
    f.write(r.content)
```

In this example, the string interpolation is managed directly with Python: the templating could have been done with LaTeX tools, but it is easier and faster for our developer to inject the dynamic content in his main programming language.[6]

We can also note that we used another resource transfer mechanism: the local image file is passed as a binary object (with a base64 encoding required by the string-based JSON API).

From this simple base example, our developer can easily inject his application data in the letter to be edited; he can then take the resulting generated PDF to return it in a web page, in his own application API and/or to save it in a persistent storage system for later use.[7]

4 The road ahead

LaTeX-on-HTTP is still a project in development; it must be enhanced to meet more developers' needs and use cases. We have already received excellent feedback, but we actively encourage and need prospective users to try it and let us know their observations and feelings.

Several subjects have been left aside in this introductory paper — or are still not properly dealt with by the current LaTeX-on-HTTP implementation:

- providing an asynchronous compilation endpoint;
- bringing alternative output modes to the PDF binary, like DVI for advanced uses or PDF.js (`mozilla.github.io/pdf.js`) for universal web embedding;
- securing and isolating the compilation jobs;
- advanced and reliable run configurations;
- caching output documents for dealing with duplicated jobs;
- caching input resources for minimizing bandwidth usage;
- discovering and managing the instance capabilities (fonts, packages, etc.).

In addition, to further aid adoption and reduce the time to first interaction, library wrappers and utilities need to be provided for the popular programming languages.

The next important work, however, is not in further development but rather writing high-level documentation and presentation material.

5 Conclusion

By bringing the runtime requirement to use LaTeX to a familiar HTTP API, we remove the main friction preventing developers from adopting LaTeX in their applications for PDF document creation.

To bolster the prospective success of LaTeX and LaTeX-on-HTTP as a reference solution for developers, we must address the cultural glass wall and knowledge barriers by:

- publicizing this usage of LaTeX in web applications and IT systems with ready-to-use demonstration code and libraries;
- publishing attractive documentation and presentations of the LaTeX-on-HTTP API.

Only then can we hope to see better typeset PDF documents in our daily applications as a result.

References

[1] G. Aye. Introducing LaTeX Base. *TUGboat* 37(3):275–276, 2016. `https://tug.org/TUGboat/tb37-3/tb117aye.pdf`

[2] S. Lang, A. Schmölzer. Noob to Ninja: The challenge of taking beginners' needs into account when teaching LaTeX. *TUGboat* 40(1):5–9, 2019. `https://tug.org/TUGboat/tb40-1/tb124lang-newbie.pdf`

[3] P. Łupkowski. Online LaTeX editors and other resources. *TUGboat* 36(1):25–27, 2015. `https://tug.org/TUGboat/tb36-1/tb112lupkowski.pdf`

[4] H. Snyder. SageMathCloud for collaborative document editing and scientific computing. *TUGboat* 38(1):44–47, 2017. `https://tug.org/TUGboat/tb38-1/tb118snyder.pdf`

[5] B. Veytsman, L. Akhmadeeva. TeX in the GLAMP world: On-demand creation of documents online. *TUGboat* 31(2):236–239, 2010. `https://tug.org/TUGboat/tb31-2/tb98veytsman-glamp.pdf`

[6] B. Veytsman, M. Shmilevich. Automatic report generation with Web, TeX and SQL. *TUGboat* 28(1):77–79, 2007. `https://tug.org/TUGboat/tb28-1/tb88veytsman-report.pdf`

[7] U. Ziegenhagen. Combining LaTeX with Python. *TUGboat* 40(2):126–128, 2019. `https://tug.org/TUGboat/tb40-2/tb125ziegenhagen-python.pdf`

◇ Yoan Tournade
Soubeyrac
Le Laussou, 47150 France
`y (at) yoantournade dot com`
`https://yoantournade.com`

[6] In a more complex case, we could have used a dedicated templating engine, such as Jinja2 (`github.com/pallets/jinja`). For more about combining LaTeX and Python, see [7].

[7] Complete demo code can be found for Python at `github.com/YtoTech/talk-TUG2020-LaTeX-on-HTTP` and in JavaScript at `github.com/YtoTech/latex-on-http-demo`.