Version 1.2

Assignment 1 – IDG2001

File processing web service

Table of Contents

In this assignment, we will create a simple web service where a user can upload files, the system will process them, and return new/processed files.

Part of the goal is to make a system where you can at a later point copy the entire project, modify the processing internals, and quickly have a new cool custom tool.

NB! Groups of 3-4 are recommended. If you want other sizes, discuss with teacher.

NB! If you would like to do a different file processing than what is suggested (E.g., file generator, invoice generator, conversion tool, etc.), discuss it with me (the teacher), and I'll probably accept it. Making the project "your own" may be more interesting. And if you have other technical solutions, feel free to suggest those as well. You have a lot of freedom here. Feel free to use it!

NB! If you use Railway.app – as used in class – and your free trail runs out, then there are a few options:

- Pay for premium – Maybe the easiest solution, but not expected.
- Use another platform. For example, Render or Netlify.
- Use one group member's account at the time.

Version 1.2

NB! The labs (and lectures to a lesser extent) are very relevant here. You get a lot for free. Use it!

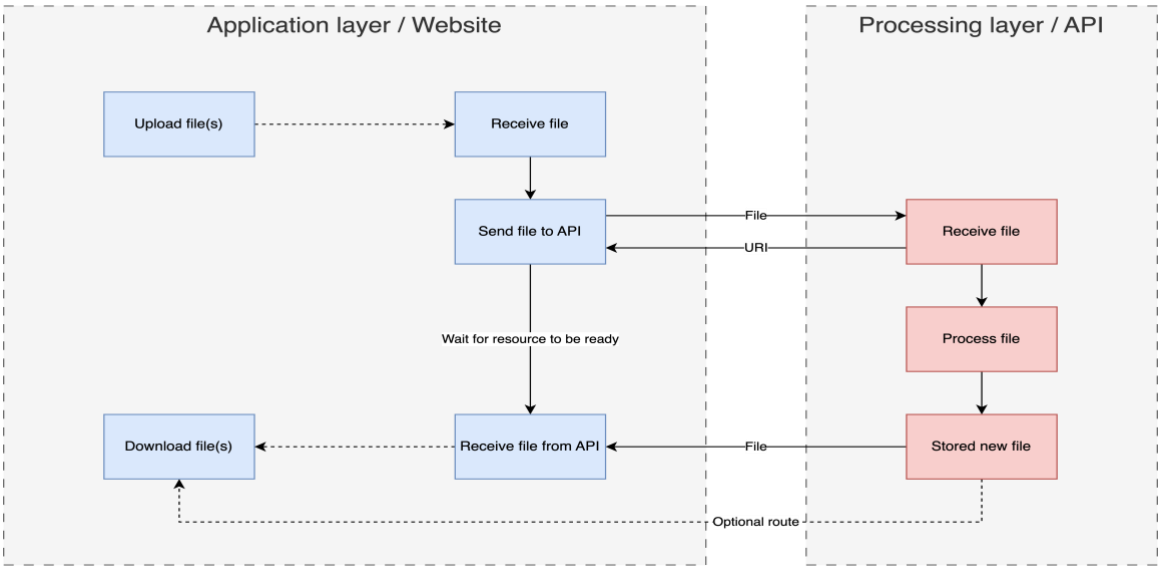
System description

I have the following files (see examples at the end of the document):

- A CSV-file of persons who should get a certificate (nor: attest).
- A Markdown template file for the certificate.
- Figures/images (0-to-many).

I want to go to your website, upload these files, and download a tar.gz file (compressed file format) of PDF files – one per person in the CSV file. If I send a CSV file with 100 names, I except 100 PDF files, with the names replaced. I should also be able to upload a single tar.gz file with the same contents.

API system



The API server should receive a tar.gz, process it, and make the new file available for download. The processing should do the following:

1. Unpack into a new folder.
2. Load CSV and Markdown file.
3. Perform replaces and store Markdown files per person (in a folder named “MD”).
4. Perform Markdown-to-PDF conversions for all files (in a folder named “PDF”).
5. Compress PDF-folder to a new tar.gz file.
6. Make it available for a GET method at its URI.

Though if you have other ways you'd like to use, just discuss with teacher, and it might be fine.

Website system

While the server is processing, the website should wait. When the file is ready, it should be downloaded to the client.

The website server should receive file or files, compress if not already compressed, send to API, receive new compressed file when ready, and download. You can choose programming language on the website part.

Other specs

The system should be made with loosely connected components which are easily moved and replaced.

- The system should preferably not be too locked to whichever cloud provider we use. I.e., we should be able to move it to another cloud provider without too much hassle.
- The processing layer/API should be easily replaceable with another. I.e., if we want to generate another type of file, we should only have to use another API.

This may be relevant for assignment two hint hint wink wink (maybe).

Code (Python)

tl;dr: Write good Python code. (I don't care as much about your JavaScript code.)

Follow the Python style guide (PEP8). (Use "flake8" or "black" to help.) Feel free to ignore rules but remember to add the ignores to your testing system (so flake8 will not fail when running).

Use type hinting – at least in some cases. Mostly so the type testing is actually testing something. Use it for simple functions and variables with only one possible datatype. (E.g., no "int | None".)

Put code in functions (and maybe classes) rather than having most of your code in the global space. That makes it easier to test and document. Add doc-strings to files/modules, classes, and functions/methods. And don't make huuuge functions.

Development pipeline

The following should be included in the development. (Configs and generated documentation should be included in the repo.)

- Tests
  - flake8
  - mypy
  - pytest
  - tox (running the above tests in multiple environments)
  - GitHub Actions running tox (on multiple OSes and Python versions.)
- Documentation
  - Descriptive README.md files in your repos' root folders.
  - Generate coverage reports, using pytest-cov or similar.
  - Generate documentation, using Sphinx or similar systems.

The above mentioned (except for the README.md point) is only for the processing layer. I do not except well-written tests, well-documented code, etc. but I except *some* tests, some documentation, etc.

Report

The report should be brief and give me an understanding of the following:

- Which significant choices you have had to make during development, what the options were, and why you made them.
- How the system works and how it is structured.
- An overview of the code (regular documentation should be in code).
- How to use the system.
- How to set up a system like this.
- How to replace the file processing with another use case.

Connect to theory/lectures where relevant. Use diagrams where relevant. Not much reward for design skills here but making it understandable is important.

Bonus points

If we find some of your deliverables lacking when grading, points will be subtracted. If you have done something a little extra, however, it can cancel subtractions. It can therefore be good to do more than expected. For example:

- Quality code tests and documentation, and high-ish test-coverage.
- Use of two cloud provides – one for the processing layer and one for the web server layer.
- Perform some testing, documentation, etc. on the website server.
- More ideas: Discuss with/email teacher.

Deliverables

- Copy of repos (as compressed files, like zip).
- Report, which can be included in repo (as a PDF, Docx or similar).

If the examination platform (Blackboard or Inspera) requires a single file, compress into a compressed file, and deliver it.

Suggested project plan

You can of course complete the pieces in whichever order and on whichever time schedule you'd like, but here are some suggestions, roughly divided into weeks. Adjust based on which lectures/labs we've completed.

Week	Work tasks
1	Set up the API test environment, doc. generation, website environment, and initialize the report (write how-to notes and documentation as you go along).
2	Make a (simple?) local version of the processing layer. First as a script, then as an API.
3	Make it work on Railway.app (and add complexity?).
4	Set up the website.
5	Finalize, test, bonus points, etc.
6	Finish report.

File examples

CSV

```
FirstName,LastName
Paul,Knutson
Barack,Obama
Erna,Solberg
```

Markdown template file

```
# Certificate of excellence

![NTNU-logo](NTNU-logo.png)


{{FirstName}} {{LastName}} have completed the course IDG2001 Cloud Technologies at (NTNU). As part of their course, they have blabla skills, lists, etc.
```

```
- SaaS, PaaS, IaaS
- Cloud infrastructure ... etc

![Signature](signature.png)

Paul Knutson, Faculty of IE, NTNU
```

Here, the files “NTNU-logo.png” and “signature.png” would have to be part of the uploaded files. {{FirstName}} would be replaced with “Paul” for person 1, “Barack” for person 2, etc.