# Cloud Oblig 1 - Report

Link to railway: https://cloudoblig1-production.up.railway.app/

## 1. Significant Choices During Development:

**Flask Framework:** We went with Flask, a widely-used and user-friendly framework, mainly because it's easy to use and flexible for web development. Since we were already a little familiar with Flask from our learning, it helped us work faster and smoother without needing to learn something new. Although we didn't check out other options, Flask's popularity and our comfort with it made it our choice for the project. Its simplicity allowed us to focus on building our application without getting confused by complicated stuff. In short, Flask was our top choice because it's recommended and we knew how to use it.

**PDF Generation:** In our project, the choice of PDF generation library was a critical decision that influenced the ease of development, deployment, and the quality of output. We evaluated several options, including WeasyPrint, WKHTMLTOPDF/pdfkit, and FPDF, each with its own set of advantages and challenges.

We initially considered WeasyPrint and WKHTMLTOPDF/pdfkit due to their popularity and robust features for generating PDFs from HTML or Markdown content. However, as we delved deeper into our deployment strategy, we encountered complications with these libraries, particularly when deploying to platforms like Railway. Integration issues and dependencies on external tools posed challenges that could potentially hinder the deployment process and introduce complexities in the application architecture.

Given these considerations, we opted for FPDF as our PDF generation solution. FPDF offered seamless integration with our Flask application and eliminated the need for external dependencies or complex setup procedures. Its lightweight nature and native support for generating PDFs from Markdown content made it an ideal choice for our project requirements.

One of the key factors that influenced our decision was FPDF's ability to produce high-quality PDF outputs with precise line breaks, consistent spacing, and well-formatted text. Despite being a simpler library compared to WeasyPrint or WKHTMLTOPDF/pdfkit, FPDF proved to be highly efficient and reliable in generating visually appealing PDF documents.

## 2. How the system works and how it is structured.

The system is a Flask-based web application designed to handle file uploads, process them, and generate PDF files based on Markdown templates and CSV data. Here's how it works and how it's structured:

1. Flask Application Setup: The system is built using Flask, a Python web framework. It sets up a Flask application and defines routes for handling file uploads and rendering HTML templates.

2. Upload Form: The '/' route renders an HTML template ("upload_form.html") containing a form for file uploads. Users can upload a .ta.gz file containing the md file, csv file and needed images

3. File Upload Handling: When files are submitted via the form, the '/upload' route receives the files. It saves the uploaded files to a temporary directory and extracts their contents.

4. File Processing: After extracting the files, the system dynamically loads Markdown and CSV files from the uploaded content. It then processes the Markdown content, performing replacements using data from the CSV file. This happens dynamically based on the csv titles.

5. PDF Generation: The modified Markdown content is converted to PDF files using the FPDF library. Each PDF corresponds to a person's data from the CSV file.

6. File Compression: Once PDF files are generated, they are compressed into a tar.gz archive.

7. File Download: The tar.gz archive is made available for download to the user. Upon successful completion, the temporary directory containing uploaded files is deleted.

8. Server Configuration: The Flask application is configured to run on a specified port, with the ability to customize the port using an environment variable.

## 4. How to Use the System:

**Upload:** To begin using the system, navigate to the web interface provided. Once there, select the desired files for upload, ensuring they are packaged within a tar.gz archive. The archive should contain a Markdown (.md), a CSV (.csv), and image files required for processing. The files can be named anything you would like. After selecting the file, simply click the "Upload" button to initiate the upload process.

**Processing:** Upon successful upload, the system automatically processes the uploaded files. It begins by extracting the contents of the tar.gz archive, allowing access to individual files. The system then proceeds to process these files, dynamically generating PDF documents based on the provided Markdown content and CSV data.

**Download:** Once the processing is complete the system compresses the generated PDF documents into a single .tar.gz archive download. Users can then download the compressed archive, which contains all the processed PDFs ready for use.

**Pytest:** NB to use pytest, you have to make sure you are in the project directory. This can be done by entering this to the terminal:

```
cd project
```

**Example files:** We have added a folder named “example-files” containing 3 different tar.gz files with different md and csv content ready to be uploaded.

## 5. Setting Up the System:

Dependencies: Before using the system, ensure that Flask and FPDF are installed. This can be done by executing the following commands in the terminal:

```
pip install Flask
pip install FPDF
```

These commands will install Flask and FPDF along with any necessary dependencies, enabling the system to function properly.

Procfile: To specify the application's entry point for deployment on platforms like Railway, a Procfile is utilized. The Procfile contains instructions for running the application. In our case, the content of the Procfile is:

```
web: python project/app.py
```

This line instructs the deployment platform to run the Flask application by executing the app.py file located in the project directory. Adjust the path accordingly based on the structure of your project.

## 6. Replacing File Processing with Another Use Case:

To use the existing file processing setup for another use case, let's imagine a situation where we need to generate certificates of participation for an online workshop. In this scenario, we'll have a CSV file containing participant names, their workshop titles, and possibly some other information. We'll also have a Markdown template file with placeholders for participant names, workshop titles, and any images we want to include.

**Example CSV File (participants.csv):**

```
FirstName,LastName,WorkshopTitle
John,Doe,Introduction to Python Programming
Jane,Smith,Data Visualization Techniques
Alice,Johnson,Machine Learning Fundamentals
```

**Example Markdown Template (certificate_template.md):**

```
# Certificate of Participation

![WorkshopLogo](workshop_logo.png)

This is to certify that {{FirstName}} {{LastName}} has
successfully completed the workshop titled {{WorkshopTitle}}.

Congratulations!

![InstructorSignature](instructor_signature.png)

Dr. Amanda Smith
Workshop Instructor
```

How to Use this Setup for Another Use Case:

1. **Prepare Input Data:**

- Create a CSV file ("participants.csv") with participant names ("FirstName", "LastName") and workshop titles ("WorkshopTitle").

- Ensure that any images referenced in the Markdown template are available and correctly named.

2. **Create a Tar Archive:**

- Package the CSV file, Markdown template, and any required images into a ".tar.gz" archive using the "tar" command (make sure it is not a directory that is packaged. it wont extract data from the tar.gz file, then another directory):

```
tar -czvf workshop_certificates.tar.gz participants.csv
certificate_template.md workshop_logo.png instructor_signature.png
```

3. **Upload the Archive:**

- Upload the generated ".tar.gz" archive ("workshop_certificates.tar.gz") to the Flask application via the file upload form.

4. **Process and Generate Certificates:**

- Upon submission, the Flask application will process the uploaded archive.

- It will extract the CSV file and Markdown template from the archive, along with any images.

- The application will then dynamically replace the placeholders in the Markdown template with participant names and workshop titles.

- Using this modified content, it will generate individual certificates in PDF format.

5. **Download Certificates:**

- Once processing is complete, the application will compress the generated PDF certificates into a new ".tar.gz" archive.

- Users can then download this archive, which contains the individual certificates for each participant.

By following these steps, the existing file processing setup can be adapted to generate certificates for various workshops or events, simply by providing a new CSV file and Markdown template customized to the specific use case.