PDF Syllabus Parser — Step-by-step Guide

Goal: Upload a syllabus PDF, extract module names, hours and syllabus details, store in MySQL, and serve via Flask (deployable to AWS).

Contents

1. Overview and project structure

Short description:
A simple web app that accepts a syllabus PDF, parses modules (name, hours, syllabus text), stores them in a MySQL database, and provides a web UI + JSON API. Built with Python, pdfplumber, Flask and MySQL.

Recommended project structure:

```
syllabus_parser/
■
■■■ app.py            # Flask backend
■■■ parser.py         # PDF parsing logic (pdfplumber + regex)
■■■ database.py       # MySQL connector functions
■■■ templates/
■   ■■■ index.html    # Simple frontend
■■■ uploads/          # Uploaded PDFs (created at runtime)
■■■ requirements.txt  # Optional - list of Python packages
```

2. Windows setup (venv + packages)

Commands (run in project folder):

```
# create folder
mkdir syllabus_parser
cd syllabus_parser

# create venv
python -m venv venv

# activate (PowerShell)
venv\Scripts\Activate.ps1

# or activate (CMD)
venv\Scripts\activate.bat

# install packages
pip install flask pdfplumber mysql-connector-python
```

Tip: If pdfplumber install needs wheel build tools, install Microsoft C++ Build Tools or use a pre-built wheel.

3. MySQL setup & schema

If you have MySQL installed (Workbench/XAMPP), create database and table:

```
CREATE DATABASE syllabus_db;
USE syllabus_db;

CREATE TABLE modules (
  id INT AUTO_INCREMENT PRIMARY KEY,
  module_name VARCHAR(255),
  hours INT,
  syllabus TEXT
);
```

Create a MySQL user or use root; remember the password for database.py.

4. Parser code (parser.py)

This uses pdfplumber to extract text and a regex to find modules formatted like 'Module 1: Name (10 Hours)'. Adjust regex for your PDFs.

```
import pdfplumber
import re

def extract_modules(pdf_path):
    text = ""
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                text += page_text + "\n"

    # Regex: adjust if your PDF uses different wording/case
    pattern = r"(Module\s+\d+:\s*(.*?)\s*\((\d+)\s*Hours\))([\s\S]*?)(?=Module\s+\d+:|$)"
    matches = re.findall(pattern, text, flags=re.IGNORECASE)

    modules = []
    for match in matches:
        _, name, hours, syllabus = match
        modules.append({
            "module_name": name.strip(),
            "hours": int(hours),
            "syllabus": syllabus.strip()
        })
    return modules

if __name__ == "__main__":
    print(extract_modules("syllabus.pdf"))
```

5. MySQL connector (database.py)

Use mysql-connector-python. Update username/password as needed.

```
import mysql.connector

def get_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="yourpassword",
        database="syllabus_db"
    )

def insert_module(name, hours, syllabus):
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("INSERT INTO modules (module_name, hours, syllabus) VALUES (%s, %s, %s)",
                   (name, hours, syllabus))
    conn.commit()
    conn.close()

def get_all_modules():
    conn = get_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM modules")
    data = cursor.fetchall()
    conn.close()
    return data
```

6. Flask app (app.py) + templates/index.html

Basic Flask app to upload PDFs and show extracted modules. Also exposes /api/modules returning JSON.

```
from flask import Flask, render_template, request, jsonify, redirect
from parser import extract_modules
from database import insert_module, get_all_modules
import os

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        file = request.files['pdf']
        if file and file.filename.lower().endswith('.pdf'):
            os.makedirs('uploads', exist_ok=True)
            path = os.path.join('uploads', file.filename)
            file.save(path)

            modules = extract_modules(path)
            for m in modules:
                insert_module(m['module_name'], m['hours'], m['syllabus'])
            return redirect('/')
    modules = get_all_modules()
    return render_template('index.html', modules=modules)

@app.route('/api/modules')
def api_modules():
    modules = get_all_modules()
    return jsonify([
        {"id": m[0], "module_name": m[1], "hours": m[2], "syllabus": m[3]}
        for m in modules
    ])

if __name__ == '__main__':
    app.run(debug=True)
```

```
<!DOCTYPE html>
<html>
<head>
  <title>PDF Syllabus Parser</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstr
</head>
<body class="p-4">
  <h2>Upload Syllabus PDF</h2>
  <form method="POST" enctype="multipart/form-data">
      <input type="file" name="pdf" class="form-control w-50" required>
      <button class="btn btn-primary mt-2">Upload & Parse</button>
  </form>

  <hr>

  <h3>Extracted Modules</h3>
  <table class="table table-bordered mt-3">
    <thead>
      <tr><th>ID</th><th>Module Name</th><th>Hours</th><th>Syllabus</th></tr>
    </thead>
    <tbody>
      {% for m in modules %}
      <tr>
```

```
        <td>{{ m[0] }}</td>
        <td>{{ m[1] }}</td>
        <td>{{ m[2] }}</td>
        <td>{{ m[3] }}</td>
      </tr>
      {% endfor %}
    </tbody>
  </table>

  <hr>
  <a href="/api/modules" target="_blank" class="btn btn-secondary">View as JSON</a>
</body>
</html>
```

7. Running & testing locally

```
1. Activate your virtual environment (Windows):
   venv\Scripts\activate

2. Ensure MySQL server is running.

3. Run the Flask app:
   python app.py

4. Open the browser at http://127.0.0.1:5000, upload a PDF and check the table.

5. Visit http://127.0.0.1:5000/api/modules for JSON output.

6. Troubleshooting:
   - If pdfplumber returns None for page.extract_text(), try PyMuPDF (fitz) as a fallback.
   - If MySQL connection fails, verify username/password/host and that MySQL accepts TCP con
```

8. Optional features to add (in order of impact)

• Add a search/filter box (by keyword or hours).
• Add user authentication so uploads are per-user.
• Store uploaded PDFs in AWS S3 and keep only metadata in DB.
• Add background job (Celery) to parse large PDFs asynchronously.
• Add tests for parser (unit tests with pytest).
• Improve parser using heuristics or spaCy for better syllabus extraction.

9. AWS deployment: EC2 + RDS + Nginx + Gunicorn (high level)

```
Simplified steps to deploy production-ready:

1. Create an AWS EC2 instance (Ubuntu; t2.micro is usually fine for testing).
2. Create an RDS MySQL instance (db.t3.micro) and note the endpoint, username, password.
3. On EC2:
   - install Python, pip, git, nginx
   - clone your repo or copy files
   - create venv, install requirements
4. Configure database.py to use RDS endpoint and credentials.
5. Use Gunicorn to run the Flask app:
   gunicorn -w 3 -b 127.0.0.1:8000 app:app
6. Configure Nginx as a reverse proxy to forward port 80 to 127.0.0.1:8000.
7. Open security groups: allow port 22 (SSH), 80 (HTTP), and restrict RDS to EC2's security
8. (Optional) Use systemd to run Gunicorn as a service and enable on boot.
9. (Optional) Use Let's Encrypt certbot to enable HTTPS with Nginx.
```

10. Checklist to track progress

```
[ ] Create project folder and venv
[ ] Install packages: flask, pdfplumber, mysql-connector-python
[ ] Create MySQL database and modules table
[ ] Implement parser.py and test on one sample PDF
[ ] Implement database.py and verify inserts
[ ] Implement app.py and templates; test upload flow
[ ] Add /api/modules and test JSON export
[ ] Optional: add search and user auth
[ ] Deploy to AWS (EC2 + RDS)
[ ] Add HTTPS and monitoring
```

Need help with any step? Tell me which step you're on and I will provide exact commands and code for it.