Getting data – Web Scraping through Program

Storing data – SQL DB

Implementing – Buttons to features

1. What kind of DB is suitable for this? SQLite
2. Table schema:
   1. First And Last Name
   2. Phone number
   3. Email
   4. Address
   5. Academic details – University, Major, Status, GPA, Graduation Date
   6. Employment pref – PT/FT, current status, schedule, prev work ex
   7. Skillset
   8. Upload the CV
   9. Gender
   10. Ethnicity
   11. Languages known
3. Form details
4. Load data from file into table
5. Automate the loading
6. API to read the data and pass on to our program
7. Build a flask code
8. Build a simple HTML page with 4 buttons 🡪 updating? Loading for first time? If yes, then take to another page showing upload or fill form, if upload, run the scraping else render the form
9. Get the input from the form. How?

Topics of Technical Documentation:

* Basic home page contains what?
* What will communicate with the Pages? API
* Once communicated, how will the data be passed back and forth? Flask and Python
* If new data, then how another page will look?
* If fill form, then how google form will be traversed?
* If upload file, then what code will run to read the data? Python code
* If updating existing data, then how will it get data from sql? Query
* After finishing, how will it update data? Update

1. How to build a server?
2. How to host a server?
3. Why do we need aws pipeline?
4. Hosting provider – AWS, Azure, Google cloud, DigitalOcean
5. Create a virtual Machine
6. Web server, DB server
7. Deployment process – Kubernetes, Docker

Understanding:

1. Create a server, DB on mySql
2. Connect to the server through sql connection
3. Build a simple page to get input from user in html
4. Pass the values through flask to our code
5. Validate and send the data to mysql

Tech Stack:

1. Programming Language - Python, A Web framework - Flask, Database – MySQL
2. Dev environment – Code editor, Git, all required libraries
3. HTML Forms
4. Flask app to communicate
5. Set up the DB
6. Frontend View
7. Test application

Building a web server and hosting an application involves several steps. Below is a general guide to help you get started. The specific details may vary depending on your application stack and hosting provider.

1. Develop Your Application:

Write the code for your web application. This may involve choosing a programming language (e.g., Python, JavaScript), a web framework (e.g., Flask, Django, Express), and any necessary libraries or dependencies.

2. Choose a Hosting Provider:

Select a hosting provider that suits your needs. Popular options include AWS, Google Cloud Platform (GCP), Microsoft Azure, Heroku, DigitalOcean, and others.

3. Set Up a Version Control System:

Use a version control system like Git to manage your codebase. Platforms like GitHub, GitLab, or Bitbucket can host your repository.

3.98.95.246

cd /home/ubuntu/test1

sudo vi /etc/systemd/system/test1.service

sudo systemctl daemon-reload

sudo systemctl start test1

sudo systemctl enable test1

curl localhost:8000

[Unit]

Description=Gunicorn instance test1

After=network.target

[Service]

User=ubuntu

Group=www-data

WorkingDirectory=/home/ubuntu/test1

ExecStart=/home/ubuntu/test1/venv/bin/gunicorn -b localhost:8000 app:app

Restart=always

[Install]

WantedBy=multi-user.target

**Tools used:**

Gunicorn – WSGI. To run web apps written in python.

Nginx – Reverse proxy server

Ec2 – Instance to host our app

Flask – Web framework

Core Python – Read the data from PDF and to read the data from form (manually) and to communicate with our MySQL DB server to perform ETL

Hosting Provider – EC2

Web server – Requests from http

Web Framework - Flask

Frontend UI 🡪 Web server 🡪 nginx 🡪 gunicorn 🡪 our Flask App 🡪SQL DB server

Training Pipeline 🡪 Deployment Pipeline

Once we connect to EC2 instance – Public IP Address

**Architecture**

Web framework Logic – Python Flask API

AWS RDS

Front end Pages

Front end HTML/CSS/Javascript pages

Python core logic

Hosting Provider (Web Server) – AWS EC2 Instance (using gunicorn / nginx)

Frontend:

* Page to get data that is manually entered by the student
* Page where a file can be uploaded
* Home page asking for student to select options
* Second page that redirects students based on upload or filling

Python core logic:

* Function to perform data scraping once the file is uploaded
* Logic to connect to SQL dB to perform ETL operations

Web Framework API:

* Manage the interaction logic between the frontend and the Python source code for the core logic
* Repository management for the entire Flask Application

Hosting/Deployment on Web server:

* Setup EC2 instance and ready the environment
* Deploy the Flask App into the Instance using any

SQL DB Management:

* Build the schema for all the data
* Monitor and review the ETL operation commands sent and received from the core Python source code