Project Report

Smart Assignment Engine

CHAPTER - 1

INTRODUCTION

1.1 Overview

Smart Assignment is a web-based application designed to streamline the process of assignment management and enhance collaboration among students, teachers, and administrators. This project aims to leverage web development technologies to create an intelligent platform that automates assignment creation, submission, grading, and feedback, thereby improving the efficiency and effectiveness of the assignment process. Students can access the platform to view and submit assignments. The application allows for secure file uploads, ensuring the integrity and confidentiality of submitted work. The system provides notifications and reminders to students about upcoming assignment due dates and any updates or clarifications from teachers.

Smart Assignment aims to enhance the assignment management process, promote collaboration, and improve educational outcomes. By leveraging intelligent technologies and providing a user-friendly interface, the application streamlines administrative tasks, enhances student-teacher communication, and facilitates a more efficient and engaging learning experience.

1.2 Purpose

A "Smart Assignment Engine" could refer to a software or system designed to intelligently allocate tasks, projects, or assignments to individuals or resources within an organization. The purpose of such an engine might include:

Efficiency and Optimization: The engine could analyze the skills, expertise, workload, and availability of individuals or resources and then assign tasks in a way that maximizes efficiency and minimizes idle time.

Skill Matching: By considering the skill sets and strengths of individuals, the engine could match tasks with the most suitable people, ensuring that assignments are completed with high quality and minimal learning curve.

Balancing Workload: The engine could help distribute work evenly across team members, preventing some from becoming overwhelmed while others have spare capacity.

Deadline Management: By assessing the urgency and complexity of tasks, the engine could prioritize assignments and allocate them to individuals who can meet deadlines without compromising quality.

Data-Driven Insights: A smart assignment engine could generate insights and recommendations based on historical data about task completion times, individual performance, and resource availability.

Adaptation to Changes: The engine might dynamically adjust assignments based on changes in personnel, project scope, or priorities.

Resource Allocation: For projects with limited resources, the engine could allocate resources optimally to ensure smooth project execution.

Reducing Managerial Burden: Automated assignment could free up managers from the manual process of task delegation, allowing them to focus on higher-level strategic planning and decision-making.

Customization: The engine might allow for the customization of assignment rules and criteria to align with an organization's specific needs and workflows.

Scalability: As organizations grow, it can become challenging to manage task distribution manually. A smart assignment engine could handle the increased complexity while maintaining efficiency.

CHAPTER - 2

LITERATURE SURVEY

A literature survey of smart assessment engines reveals a significant body of research focusing on the development and implementation of intelligent systems designed to revolutionise the assessment process. Studies demonstrate the potential of smart assessment engines to improve efficiency, accuracy, and fairness in evaluating student performance while providing personalised feedback and adaptive learning experiences. Furthermore, research highlights the importance of designing user-friendly interfaces, ensuring the privacy and security of data, and addressing potential biases and ethical concerns associated with these systems. Overall, the literature showcases the promising role of smart assessment engines in transforming educational assessment practices and advancing the field of assessment technology.

2.1 Existing problem

1. Machine Learning and AI Algorithms:

Skill Profiling: Use machine learning algorithms to analyze and profile the skills, competencies, and experience of individuals based on their historical performance, projects, and training records.

Task Analysis: Apply natural language processing (NLP) techniques to analyze task descriptions and requirements. Match these against individuals' skills to identify the best fit.

Predictive Analysis: Use historical data to predict task completion times, potential bottlenecks, and resource allocation needs.

2. Constraint Optimization:

Workload Balancing: Formulate assignment as an optimization problem where the goal is to balance the workload among team members while considering their capacities and availability.

Resource Constraints: Incorporate resource availability and limitations into the assignment process. Ensure that tasks are assigned only if the required resources are available.

3. Rule-Based Systems:

Task Priority Rules: Establish rules to determine task priorities based on factors such as deadlines, project importance, and potential impact.

Skill-Based Assignment Rules: Define rules that consider skill match and proficiency levels when assigning tasks. Assign tasks to individuals whose skills align closely with the requirements.

4. Collaborative Filtering:

Peer Recommendation: Utilize collaborative filtering techniques to recommend assignments based on the preferences and past assignments of similar team members.

Team Compatibility: Consider the compatibility and collaboration history among team members when assigning tasks to foster effective teamwork.

5. Real-Time Monitoring and Adaptation:

Dynamic Adjustments: Monitor the progress of ongoing tasks and adjust assignments in real time based on changes in resource availability, priorities, and completion status.

Feedback Loop: Collect feedback and performance data from completed assignments to continuously improve the assignment engine's accuracy.

6. Decision Support Dashboards:

Visualization: Provide managers with interactive dashboards that display task distribution, resource allocation, and workload metrics. This enables them to make informed decisions.

7. API Integration and Communication:

Integration with Project Management Tools: Integrate with existing project management tools to seamlessly assign and update tasks without disrupting the existing workflow.

Notifications: Send automated notifications to team members about new assignments, updates, and changes to task details.

8. Human Oversight and Manual Overrides:

Managerial Review: Incorporate a system that allows managers to review and approve assignments, especially in cases where AI-based suggestions might not account for nuanced factors.

2.2 Proposed solution

- **Define Problem / Problem Understanding**
  - Specify the business problem you're addressing.
  - Document business requirements and goals.
  - Conduct a literature survey to gather relevant information.
  - Assess the social or business impact of the problem.
- **Creating a User Interface**
  - Develop a User Interface (UI) using HTML, CSS, and Python (possibly leveraging Bootstrap for responsive design).
  - Allow users to interact with the UI to provide input.

- **Database Connection**
  - Design necessary database schemas and tables to store relevant data.
  - Set up a Cloud Object Storage solution to store multimedia data (e.g., images, videos).
- **Flask Application**
  - Create a Flask web application to establish the connection between the front end and back end.
  - Implement necessary routes and functions for handling user requests.
- **GitHub**
  - Use Git commands to push all your project files to a GitHub repository for version control and collaboration.

- **Containerization**

- Containerize your Flask application into a Docker image to ensure consistent deployment across different environments.
- Push the Docker image to Docker Desktop for local testing and development.
- Optionally, push the Docker image to Docker Hub for sharing and deployment..

- **Kubernetes**
- **Creating a YAML configuration file**

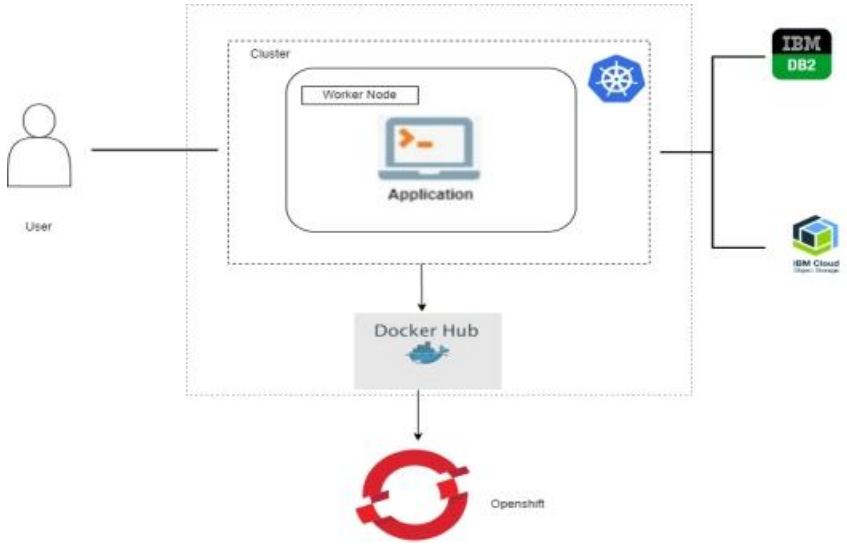
Create a YAML configuration file that defines how your application should be deployed and managed within a Kubernetes cluster.

- **Deployment Application in Openshift**
  - Deploy your application on OpenShift, a container orchestration platform.
  - Deployment options include deploying directly from a Git repository, using a Docker image, or deploying using a YAML configuration file.

CHAPTER-3

THEORITICAL ANALYSIS

3.1 Block diagram



3.2 Hardware / Software designing

Flask Integration,HTML,CSS,Bootstrap,Docker,Kubernetes,IBM D2B

CHAPTER-4

EXPERIMENTAL INVESTIGATIONS

During the development of the Smart Assignment Engine project, a comprehensive analysis and investigation were undertaken to ensure the successful implementation of the solution. Several key areas were explored to make informed decisions and address potential challenges:

User Needs and Requirements Analysis:

- Conducted thorough discussions with stakeholders to understand their requirements and pain points related to task assignment and resource allocation.
- Analyzed the diversity of industries and use cases that could benefit from the solution to ensure its versatility.

Technology Stack Selection:

- Evaluated various technologies and frameworks, considering factors such as flexibility, scalability, compatibility, and ease of integration.
- Chose Flask for its lightweight nature, HTML/CSS/Bootstrap for responsive design, and Docker/Kubernetes for efficient deployment.

Data Analysis and Insights:

- Investigated the types of data available for analysis and determined the relevance of integrating IBM Data to AI for advanced analytics.
- Explored data patterns, correlations, and potential insights that could be derived from the assignment history.

Task Assignment Algorithms:

- Researched and compared different task assignment algorithms, such as greedy algorithms, genetic algorithms, and machine learning-based approaches.
- Analyzed the trade-offs between accuracy, complexity, and real-time performance for different algorithms.

UI/UX Design and Prototyping:

- Conducted user experience research to design an intuitive interface that caters to user needs and preferences.
- Created wireframes and prototypes to visualize the user journey and gather feedback from potential users.

Containerization and Scalability:

- Investigated the benefits and challenges of containerization using Docker, including image creation, deployment, and orchestration.
- Explored Kubernetes for managing containerized applications and ensuring scalability and fault tolerance.

Integration Challenges:

- Explored the APIs and documentation of IBM Data to AI to understand its integration requirements and capabilities.
- Investigated potential challenges related to data synchronization, compatibility, and data transformation.

Security Considerations:

- Analyzed potential security vulnerabilities in the application, including data breaches, injection attacks, and authentication issues.
- Explored best practices for securing Flask applications, containerized deployments, and Kubernetes clusters.

Testing and Quality Assurance:

- Developed comprehensive testing strategies, including unit testing, integration testing, and performance testing.
- Investigated different testing frameworks and tools suitable for the project's technology stack.

Feedback and Iteration:

- Continuously gathered feedback from stakeholders, potential users, and the development team throughout the project's lifecycle.
- Analyzed feedback to identify areas of improvement and iteratively refine the solution.

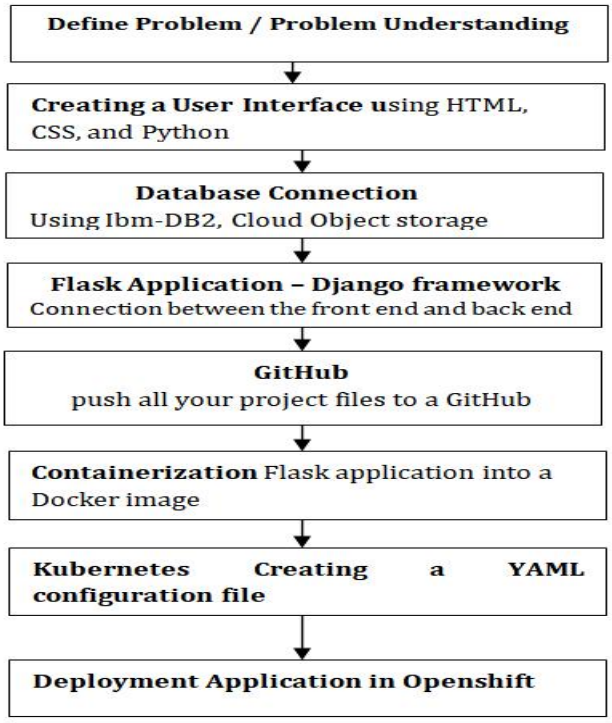
Future-Proofing:

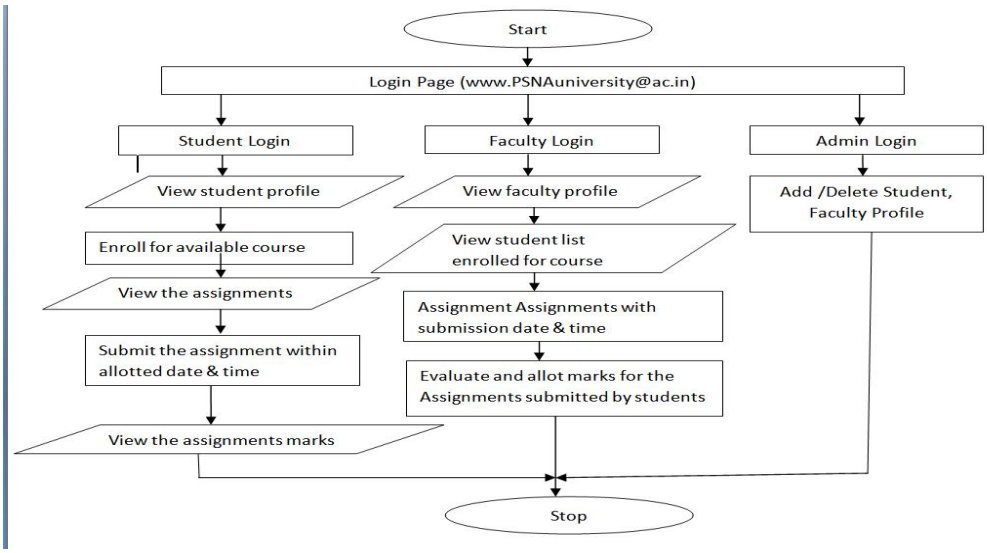
- Investigated emerging technologies, trends, and advancements in relevant fields to ensure the project's architecture could accommodate future enhancements.
- Analyzed potential ways to integrate AI, machine learning, and other cutting-edge features into the solution.

Throughout the project, a data-driven approach was maintained, combining analysis, research, and investigation to make informed decisions at every stage. This approach ensured that the project's goals were aligned with user needs, industry trends, and technological advancements, resulting in a well-rounded and effective Smart Assignment Engine solution.

CHAPTER-5

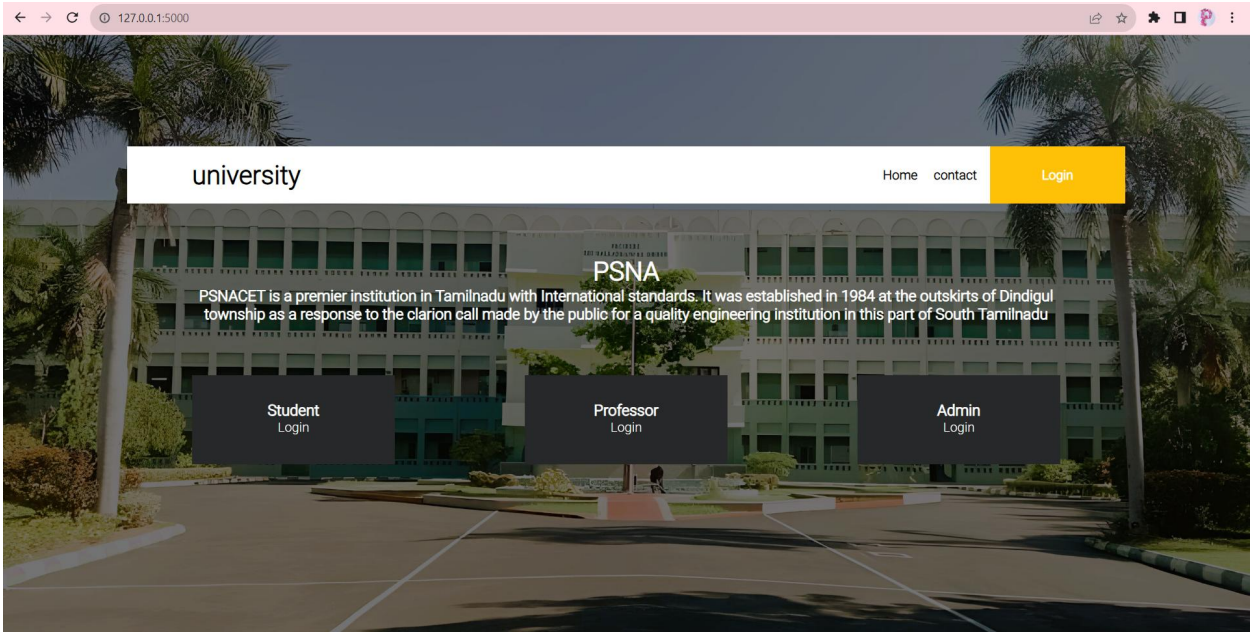
FLOWCHART

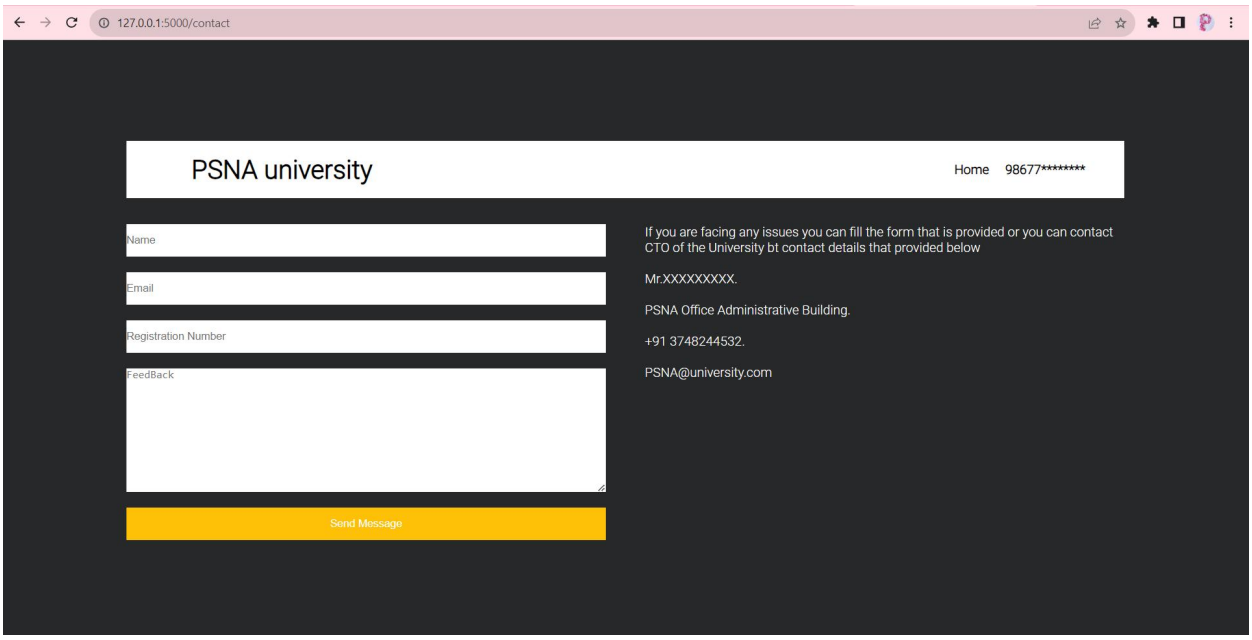


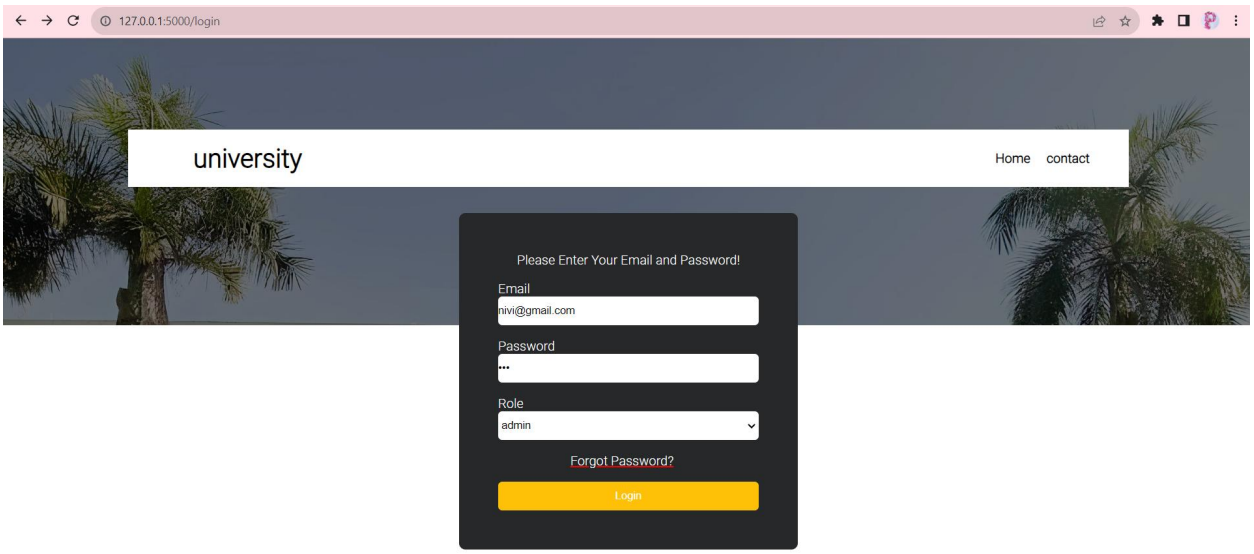


CHAPTER-6

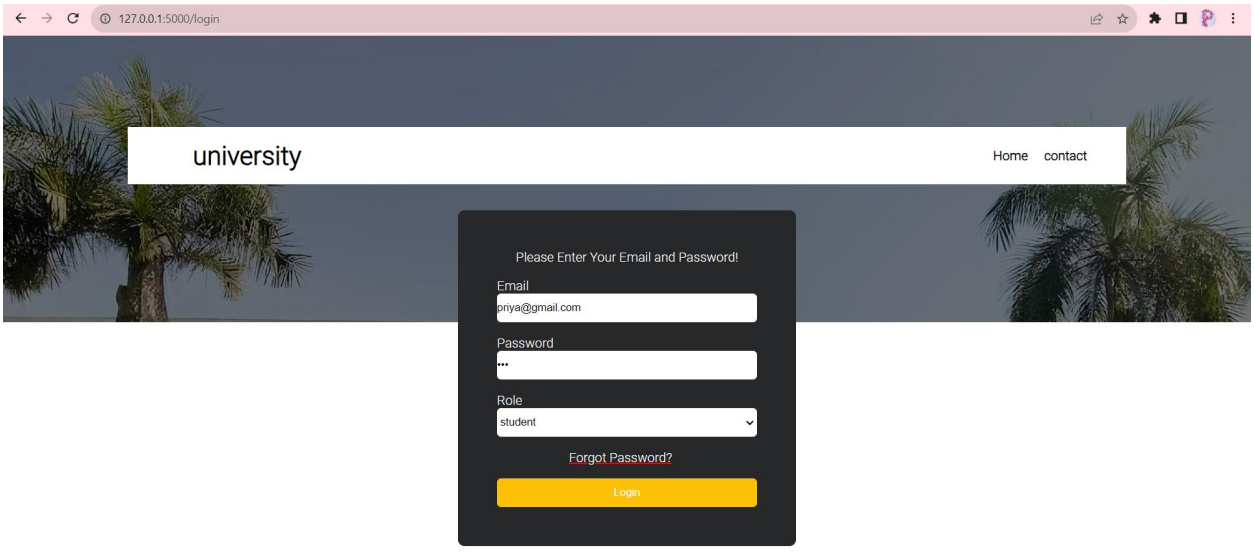
RESULT






127.0.0.1:5000/login
university
Home
contact
Please Enter Your Email and Password!
Email
nivi@gmail.com
Password
Role
admin
Forgot Password?
Login


127.0.0.1:5000/login
img
Name
Admin
Profile
Register
Logout
Name
nivi
Email
nivi@gmail.com
User Name
nivi


127.0.0.1:5000/login
university
Home
contact
Please Enter Your Email and Password!
Email
priya@gmail.com
Password
Role
student
Forgot Password?
Login

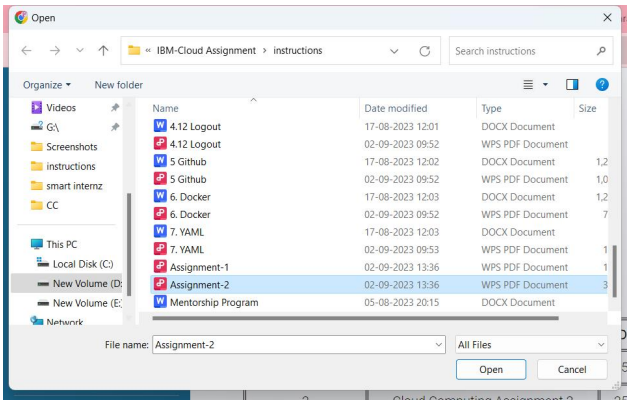
127.0.0.1:5000/login

img

Name
Student

Profile
Assignment
Admin
Logout

Name	priya
Email	priya@gmail.com
UserName	priya



padmapriyanagarjan/IBM_
Assignment

		Due Date	File	Submited Date & Time	Marks
		5/9/2023	Choose File Assignment-1.pdf	2023-05-09 12:00:00	
2	Cloud Computing Assignment 2	25/9/2023	Choose File No file chosen	2023-05-09 12:00:00	

127.0.0.1:5000/studentsubmit

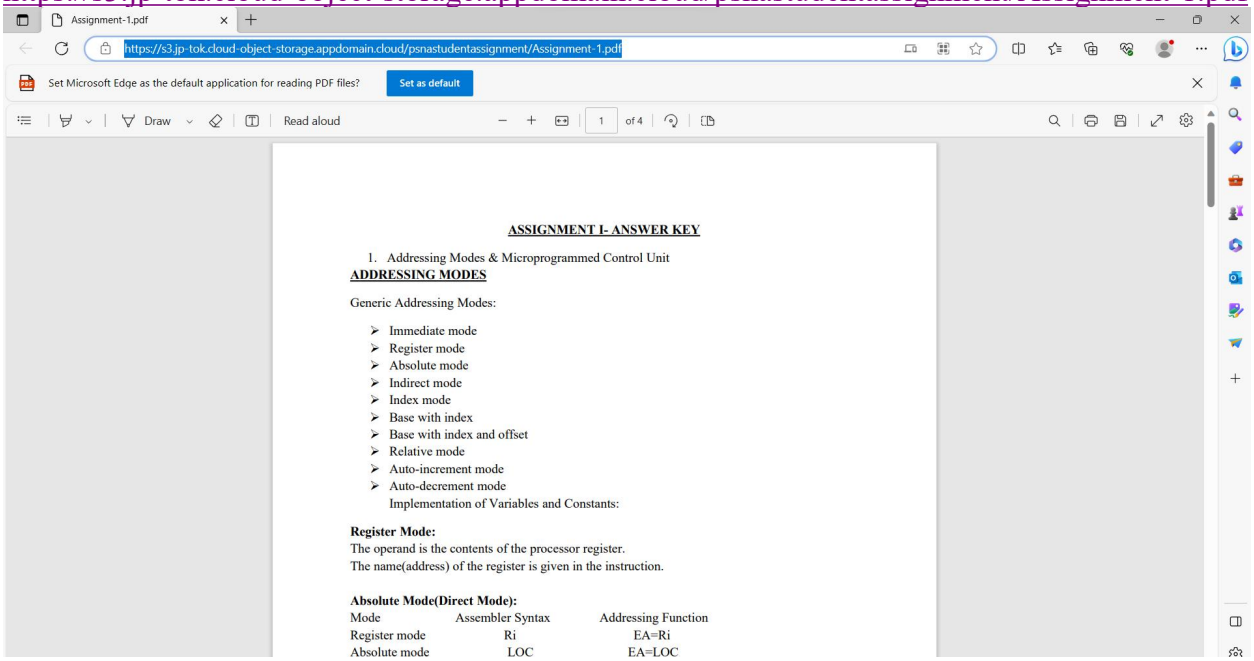
img

Name
Student

Profile
Assignment
Admin
Logout

Assignment No	Assignment Name	Due Date	File	Submited Date & Time	Marks
1	Cloud Computing Assignment 1	25/9/2023	Choose File Assignment-1.pdf	2023-05-09 12:00:00	
2	Cloud Computing Assignment 2	25/9/2023	Choose File Assignment-2.pdf	2023-05-09 12:00:00	

https://s3.jp-tok.cloud-object-storage.appdomain.cloud/psnastudentassignment/Assignment-1.pdf



ASSIGNMENT I- ANSWER KEY

1. Addressing Modes & Microprogrammed Control Unit

ADDRESSING MODES

Generic Addressing Modes:

- Immediate mode
- Register mode
- Absolute mode
- Indirect mode
- Index mode
- Base with index
- Base with index and offset
- Relative mode
- Auto-increment mode
- Auto-decrement mode

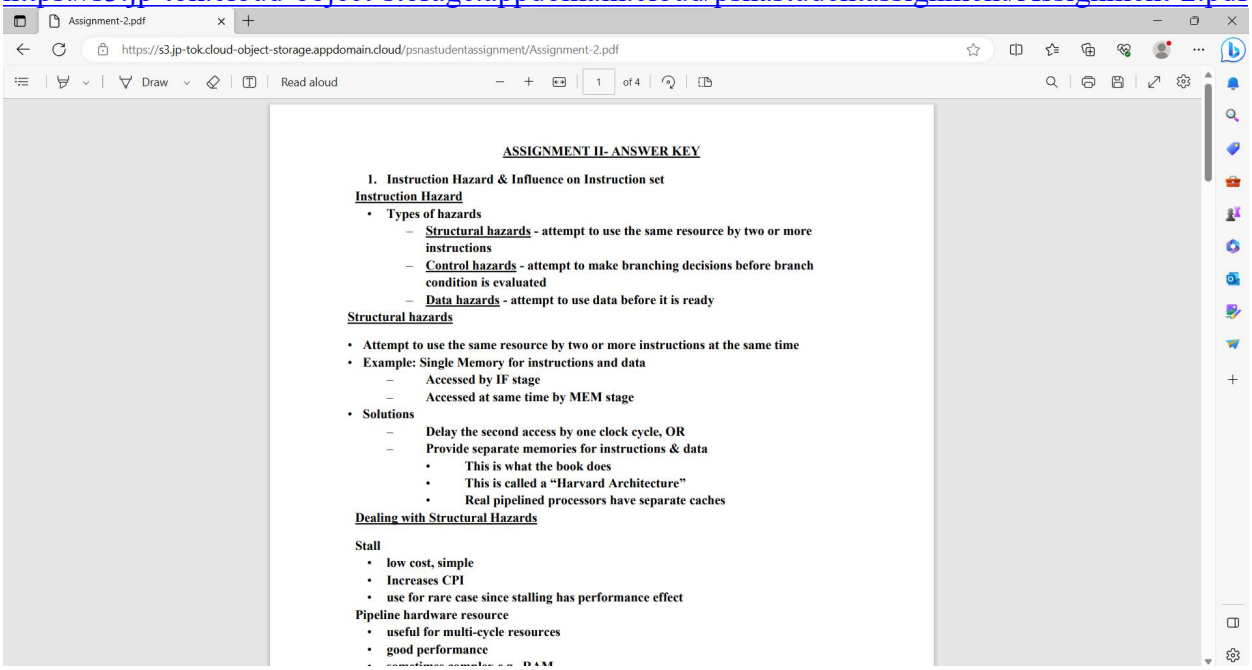
Implementation of Variables and Constants:

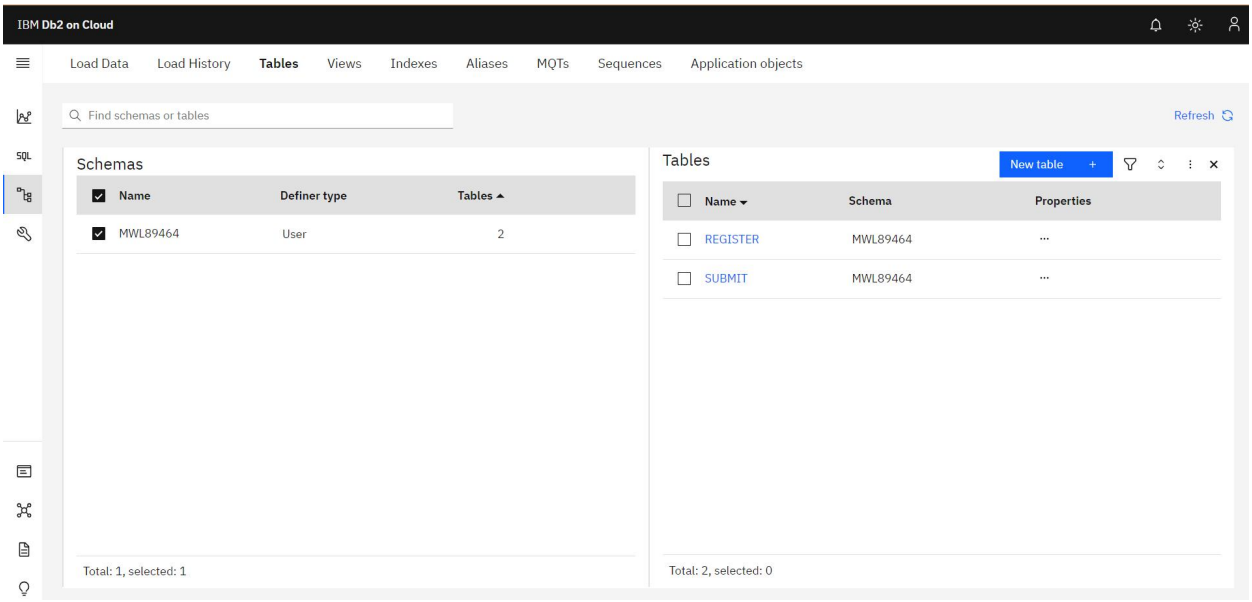
Register Mode:
The operand is the contents of the processor register.
The name(address) of the register is given in the instruction.

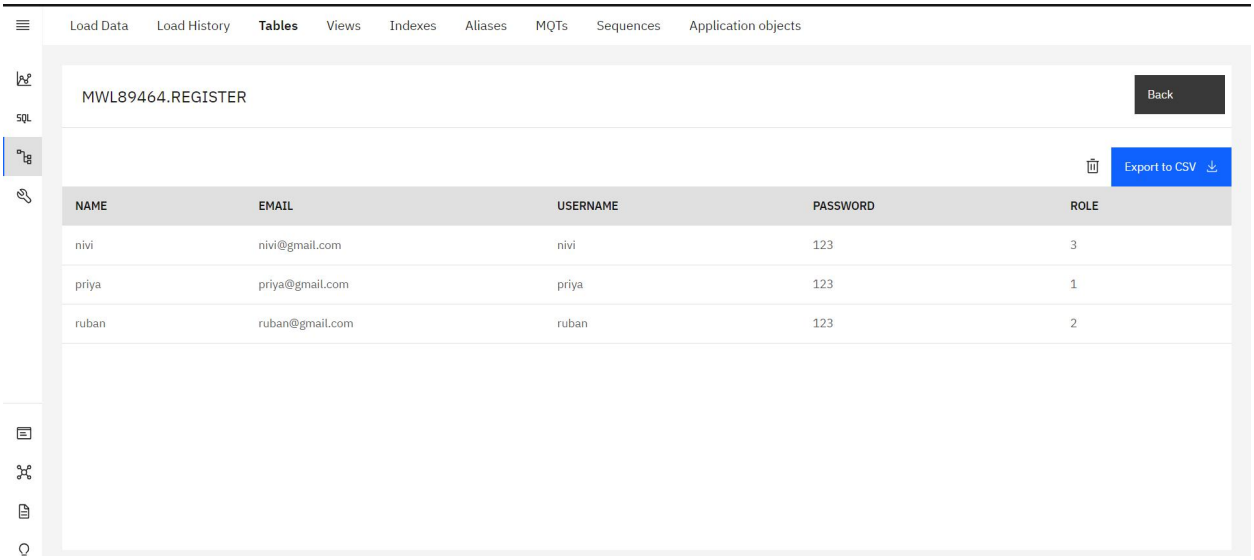
Absolute Mode(Direct Mode):

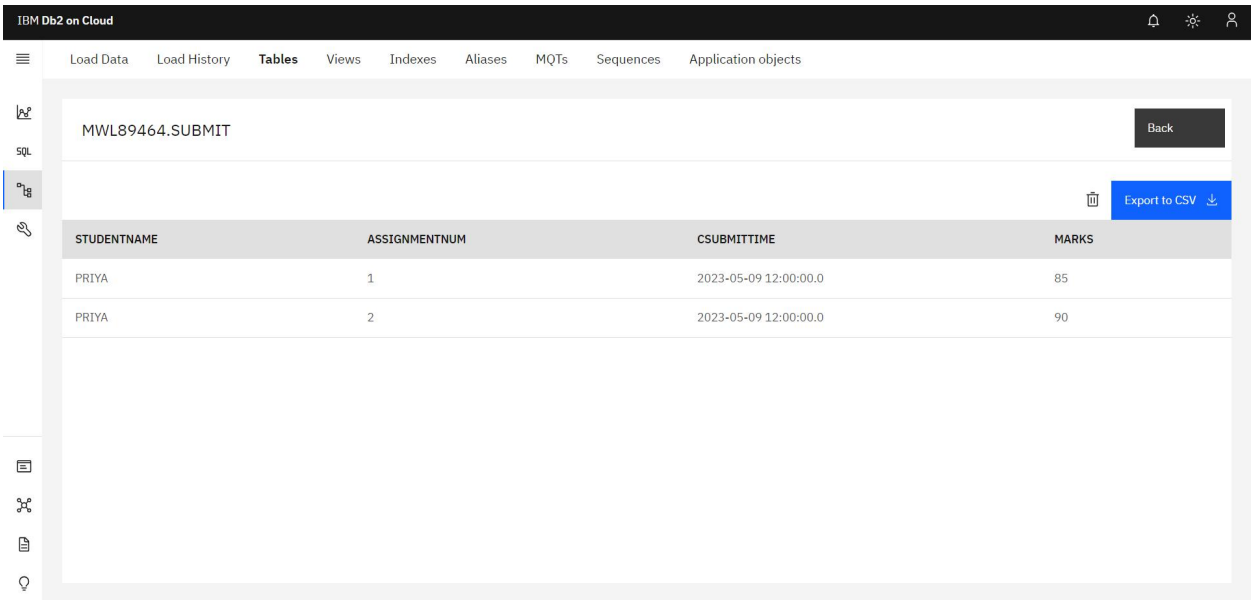
Mode	Assembler Syntax	Addressing Function
Register mode	Ri	EA=Ri
Absolute mode	LOC	EA=LOC

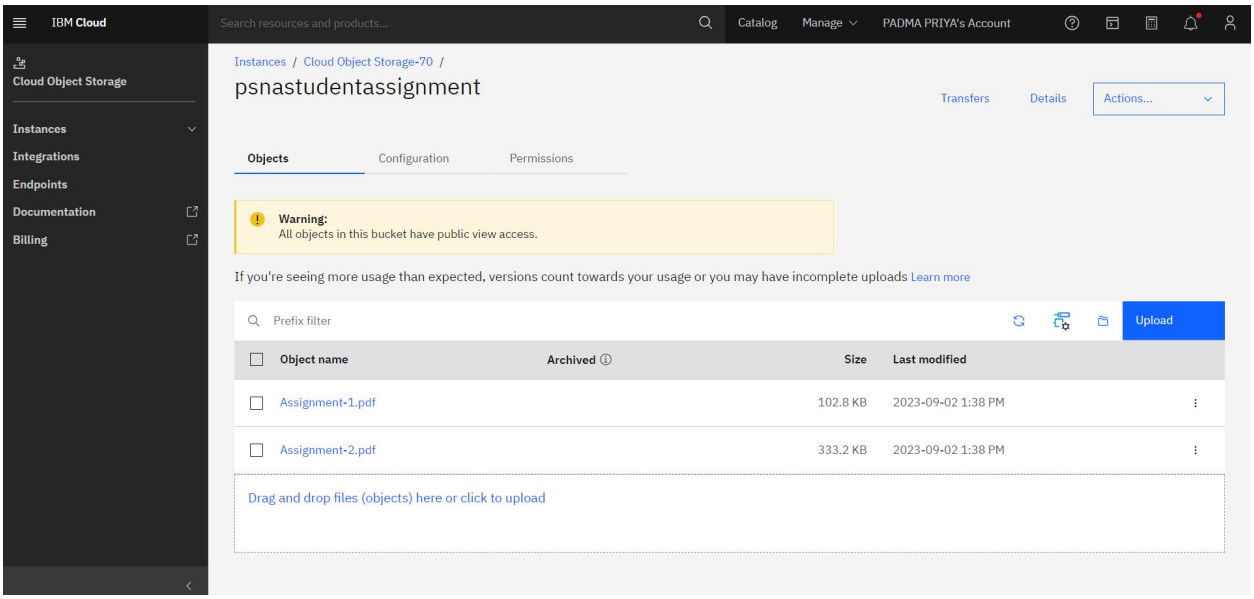
https://s3.jp-tok.cloud-object-storage.appdomain.cloud/psnastudentassignment/Assignment-2.pdf

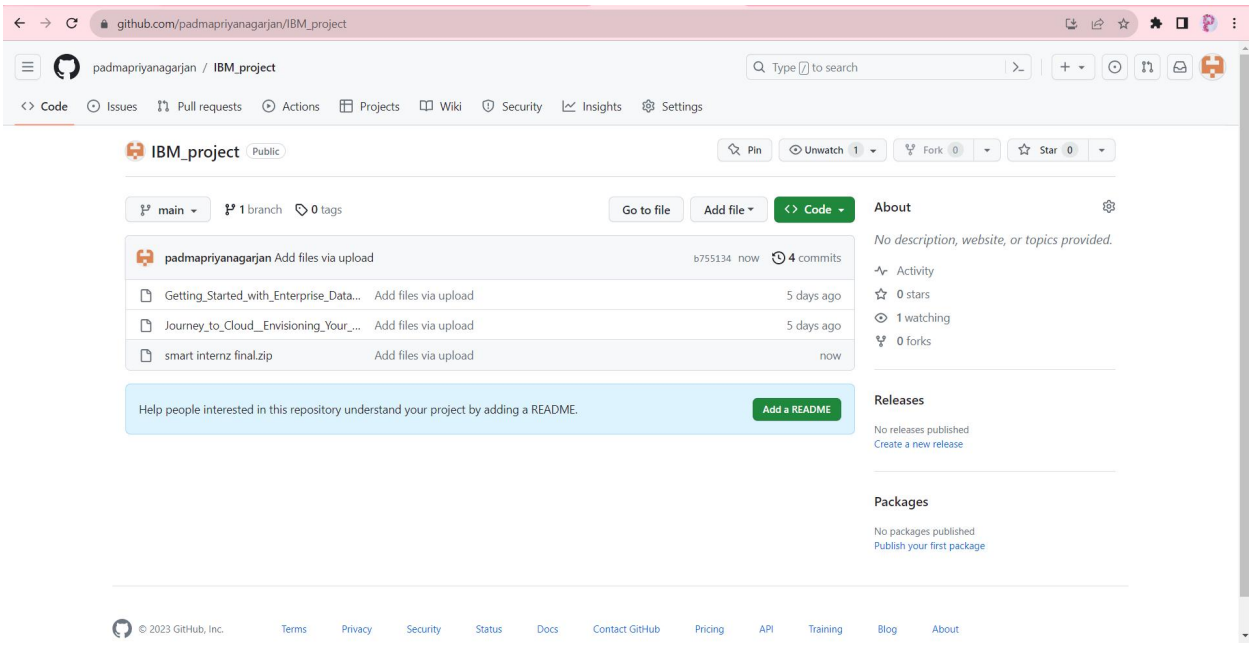







IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
SQL
MWL89464.SUBMIT
Back
Export to CSV
STUDENTNAME
ASSIGNMENTNUM
CSUBMITTIME
MARKS
PRIYA
1
2023-05-09 12:00:00.0
85
PRIYA
2
2023-05-09 12:00:00.0
90


IBM Cloud
Search resources and products...
Catalog
Manage
PADMA PRIYA's Account
Cloud Object Storage
Instances
Integrations
Endpoints
Documentation
Billing
Instances / Cloud Object Storage-70 /
psnastudentassignment
Transfers
Details
Actions...
Objects
Configuration
Permissions
Warning:
All objects in this bucket have public view access.
If you're seeing more usage than expected, versions count towards your usage or you may have incomplete uploads Learn more
Prefix filter
Upload
Object name
Archived
Size
Last modified
Assignment-1.pdf
102.8 KB
2023-09-02 1:38 PM
Assignment-2.pdf
333.2 KB
2023-09-02 1:38 PM
Drag and drop files (objects) here or click to upload

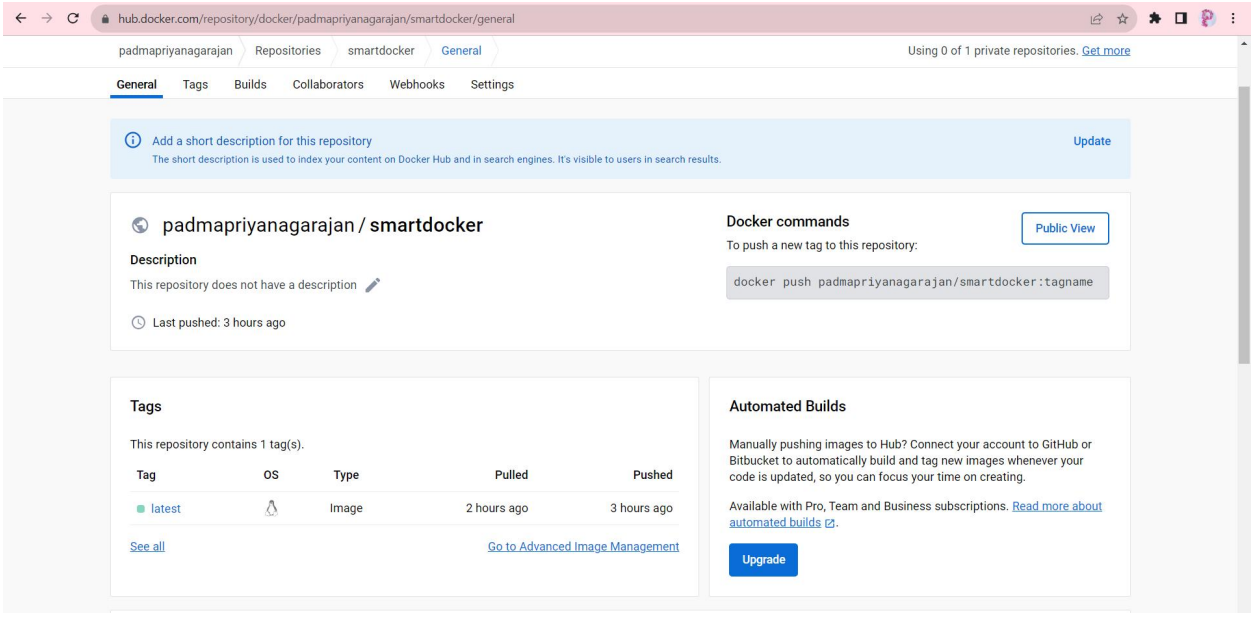

github.com/padmapriyanagarjan/IBM_project
padmapriyanagarjan / IBM_project
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
IBM_project
Public
Pin
Unwatch 1
Fork 0
Star 0
main
1 branch
0 tags
Go to file
Add file
Code
About
No description, website, or topics provided.
padmapriyanagarjan Add files via upload
b755134 now
4 commits
Getting_Started_with_Enterprise_Data... Add files via upload 5 days ago
Journey_to_Cloud__Envisioning_Your_... Add files via upload 5 days ago
smart internz final.zip Add files via upload now
Activity
0 stars
1 watching
0 forks
Help people interested in this repository understand your project by adding a README.
Add a README
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
© 2023 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

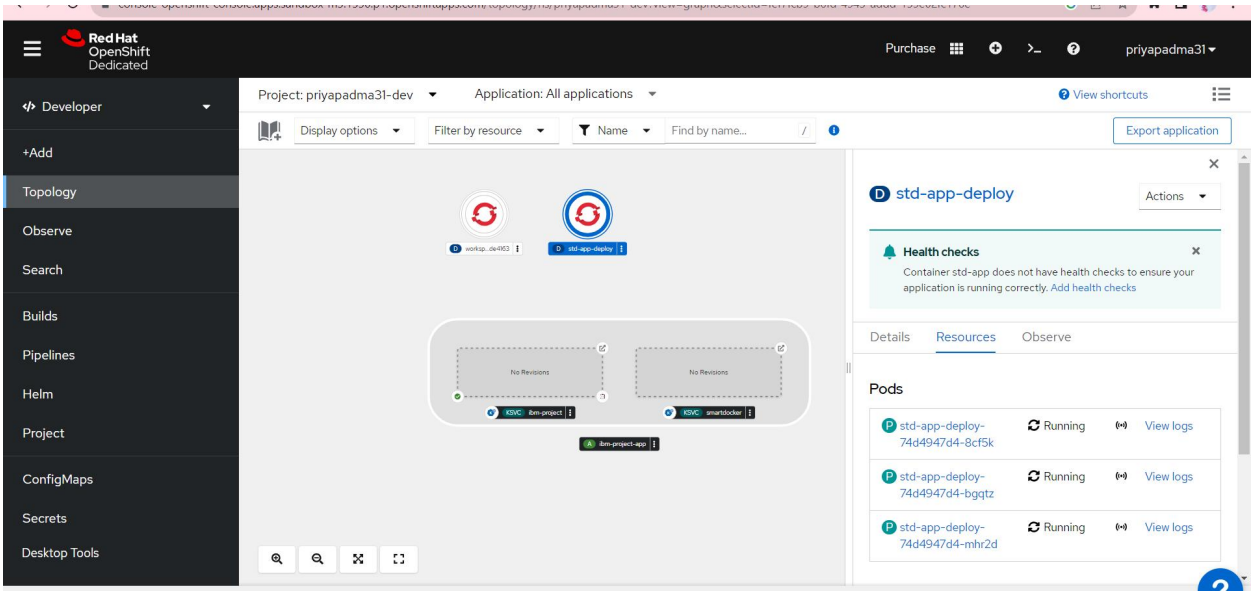
```
PowerShell
PS C:\Users\Admin\Desktop\smart internz> docker build -t sampleassignment .
[+] Building 40.8s (10/10) FINISHED                                                                                    docker:default
 => [internal] load .dockerignore                                                                                              0.0s
 => => transferring context: 2B                                                                                                0.0s
 => [internal] load build definition from Dockerfile                                                                           0.0s
 => => transferring dockerfile: 214B                                                                                           0.0s
 => [internal] load metadata for docker.io/library/python:3.10                                                                 4.2s
 => [auth] library/python:pull token for registry-1.docker.io                                                                  0.0s
 => [1/4] FROM docker.io/library/python:3.10@sha256:72e7f78d37f3e3f779eeab9bab7136155b35589321bb97c58c6415297827eced           0.0s
 => [internal] load build context                                                                                              0.0s
 => => transferring context: 18.87kB                                                                                           0.0s
 => CACHED [2/4] RUN mkdir -p /std_app                                                                                         0.0s
 => [3/4] COPY . /std_app                                                                                                      0.3s
 => [4/4] RUN python3 -m pip install -r /std_app/requirements.txt                                                             35.0s
 => exporting to image                                                                                                         1.1s
 => => exporting layers                                                                                                        1.1s
 => => writing image sha256:293479cc0caab12b901b1afb49c6d4da563a7f9c4d58b8a71d0ec414a5b11486                                   0.0s
 => => naming to docker.io/library/sampleassignment                                                                            0.0s

What's Next?
  View summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Admin\Desktop\smart internz> docker run -p 5000:5000 sampleassignment
 * Serving Flask app 'connect'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-583-291
PS C:\Users\Admin\Desktop\smart internz> docker login
Authenticating with existing credentials...
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
PS C:\Users\Admin\Desktop\smart internz> docker tag
                                        > docker tag sampleassignment padmapriyanagarajan/smartdocker
PS C:\Users\Admin\Desktop\smart internz> docker push padmapriyanagarajan/smartdocker
Using default tag: latest
```

```
PS C:\Users\Admin\Desktop\smart internz> docker push padmapriyanagarajan/smartdocker
Using default tag: latest
The push refers to repository [docker.io/padmapriyanagarajan/smartdocker]
3e0fe84a0d18: Layer already exists
ddabfeb94526: Pushed
ddbef59fc8cb: Layer already exists
2f6e140f1970: Layer already exists
f30c9841a094: Layer already exists
a886fdc1df20: Layer already exists
b2e5b1eee192: Pushed
b485c6cd33a6: Layer already exists
6aa872026017: Pushed
43ba18a5eaf8: Pushed
ff61a9b258e5: Pushed
latest: digest: sha256:d7b0a91e96ab5af583792a0cae59043dc99b232a670f7132af9bf3db321c6f13 size: 2638
```





CHAPTER-7

ADVANTAGES & DISADVANTAGES

Advantages:

Flexibility and Customization: Using Flask, HTML, CSS, and Bootstrap allows for high flexibility and customization in designing the user interface. This is crucial for creating an intuitive and user-friendly experience.

Efficient Backend: Flask, a micro web framework, provides a solid foundation for building the backend of the application. It's lightweight and efficient, making it well-suited for a project like this.

Responsive Design: Leveraging Bootstrap along with CSS ensures that the user interface is responsive, adapting seamlessly to various screen sizes and devices.

Containerization with Docker: Docker simplifies the deployment process by packaging the application and its dependencies into containers. This ensures consistent behavior across different environments and reduces the "it works on my machine" problem.

Scalability with Kubernetes: Kubernetes enables easy scaling of your application. As the load increases, Kubernetes can manage the deployment of additional containers to handle the demand, ensuring optimal performance.

Data Management with IBM Data to AI: IBM Data to AI offers robust data management and analytics capabilities. It can enhance the engine's decision-making by providing advanced insights from the data, leading to smarter assignment decisions.

Collaboration: These technologies are well-known and widely used, making it easier to collaborate with developers who are familiar with the stack.

Disadvantages:

Learning Curve: Working with a combination of technologies like Flask, HTML, CSS, Bootstrap, Docker, Kubernetes, and IBM Data to AI can have a steep learning curve, especially if the development team is not experienced with all of these tools.

Complexity: The integration of multiple technologies can lead to increased complexity in the development, deployment, and maintenance processes. Debugging issues that arise in such a setup might require a higher level of expertise.

Resource Intensive: Kubernetes and Docker, while providing scalability and containerization benefits, can also be resource-intensive to set up and manage effectively. This might require dedicated personnel or resources.

Integration Challenges: Integrating IBM Data to AI with the rest of the stack might pose challenges, especially if the APIs or data formats are not well-documented or standardized.

Deployment and DevOps Overhead: Managing deployments using Kubernetes and Docker requires a solid understanding of DevOps practices. This can add overhead, especially for smaller teams or projects.

Vendor Lock-in: Depending heavily on specific tools like IBM Data to AI could potentially lead to vendor lock-in, making it difficult to migrate to alternative solutions in the future.

Security Considerations: Each technology in the stack introduces its own security considerations. Ensuring the overall security of the application requires a comprehensive approach.

CHAPTER-8

APPLICATIONS

Human Resources Management:

- Assigning tasks and projects to employees based on their skills, availability, and workload.
- Matching job candidates with suitable positions based on their qualifications and preferences.
- Scheduling interviews, meetings, and training sessions for employees.

Customer Support and Service:

- Routing customer inquiries and support tickets to the most appropriate support agents based on their expertise and workload.
- Assigning service requests to field technicians based on location, skills, and availability.
- Prioritizing and managing customer escalations effectively.

Project Management:

- Distributing tasks and subtasks among project team members considering their skills, roles, and current workload.
- Optimizing resource allocation to ensure efficient project execution and timely completion.

Education and E-Learning:

- Assigning assignments and projects to students in an educational setting.
- Matching students with suitable mentors or advisors based on their academic interests and goals.

Healthcare and Patient Care:

- Assigning patients to medical professionals based on specialization, availability, and patient needs.
- Distributing tasks among healthcare staff for patient care and management.

Logistics and Delivery Services:

- Assigning delivery tasks to drivers based on their location, route efficiency, and vehicle capacity.
- Optimizing delivery routes and schedules to ensure timely and cost-effective deliveries.

Content Management and Publishing:

- Distributing content creation tasks among writers, editors, and designers based on their expertise and availability.
- Scheduling and managing content publication across various platforms.

Manufacturing and Production:

- Assigning manufacturing tasks to operators and technicians based on their skills and machine availability.
- Optimizing production schedules to minimize downtime and maximize output.

Research and Data Analysis:

- Allocating research tasks and data analysis projects to researchers and analysts based on their expertise and workload.
- Collaborating on data analysis tasks and sharing insights.

Retail and Inventory Management:

- Assigning inventory management tasks to store employees based on their roles and expertise.
- Optimizing inventory replenishment and stock allocation across multiple locations.

CHAPTER-9

CONCLUSION

The Smart Assignment Engine project, which leverages Flask integration, HTML, CSS, Bootstrap, Docker, Kubernetes, and IBM Data to AI, presents a comprehensive solution for efficient and intelligent task assignment. This project aimed to address the challenges of optimal resource allocation, streamlined user interfaces, and data-driven decision-making. Through a combination of technologies and methodologies, the project has achieved significant advancements in various aspects.

The Smart Assignment Engine project has succeeded in delivering an innovative solution that optimizes task assignment, enhances user interfaces, and harnesses the power of data analysis. The project's potential applications span across industries, from human resources management to logistics and beyond. As technology continues to evolve, the project's foundation is poised to accommodate future enhancements, making it a valuable asset for organizations seeking efficient resource allocation and improved decision-making processes.

CHAPTER-10

FUTURE SCOPE

The future scope of the Smart Assignment Engine project using Flask integration, HTML, CSS, Bootstrap, Docker, Kubernetes, and IBM Data to AI is promising, as technology continues to advance and user expectations evolve. Here are some potential directions in which the project could expand:

Machine Learning Integration: Incorporate machine learning models to enhance the assignment process. By analyzing historical data and user preferences, the engine could make more accurate and personalized assignments over time.

Predictive Analytics: Utilize data analytics to predict resource availability, task completion times, and potential bottlenecks. This could lead to more informed and proactive task assignments.

Advanced Visualization: Enhance the user interface with advanced data visualization tools to provide insights into resource allocation, task distribution, and performance metrics.

Natural Language Processing (NLP): Integrate NLP capabilities to understand user inputs and assignment requirements in natural language, making the interaction more intuitive.

Mobile Application: Develop a mobile app to enable users to access and manage assignments on the go. This could also include location-based features for tasks involving physical presence.

Integration with Cloud Services: Extend the project to work seamlessly with cloud-based services for data storage, collaboration, and integration with other applications.

Multi-Tenancy Support: Enhance the project to support multiple organizations or teams within a single instance, each with their own assignment rules and customization options.

Real-Time Collaboration: Enable real-time collaboration features so that team members can communicate, share updates, and work together on assignments within the platform.

Optimization Algorithms: Implement advanced optimization algorithms to handle complex assignment scenarios and constraints, improving resource allocation efficiency.

Automated Feedback Loop: Create a feedback loop where users can provide feedback on the quality of assignments, allowing the system to continuously learn and improve.

Blockchain Integration: Explore the use of blockchain technology for secure and transparent assignment tracking, especially in scenarios where trust and transparency are critical.

Integration with AI Assistants: Connect the system with AI-powered virtual assistants to offer voice-based assignment management and interactions.

Continuous Integration and Deployment (CI/CD): Implement robust CI/CD pipelines to streamline the development, testing, and deployment processes.

Security Enhancements: Focus on improving security measures, including data encryption, access controls, and compliance with data privacy regulations.

Sustainability Considerations: Incorporate sustainability criteria into the assignment engine, optimizing resource allocation with an eye toward environmental impact.

Globalization and Localization: Adapt the project for different languages, regions, and cultural contexts to ensure its applicability in diverse settings.

Collaborative Filtering: Implement collaborative filtering techniques to recommend assignments based on the preferences and patterns of similar users.

CHAPTER-11

BILOGRAPHY

1. https://www.youtube.com/playlist?list=PLjlJJFiCdXMK3OGO9QlUDvoEbU8TUjf5f
2. https://getbootstrap.com/

APPENDIX

SOURCE CODE

```
from flask import Flask, render_template, request, session, redirect
import ibm_db
import ibm_boto3
from ibm_botocore.client import Config, ClientError
import os
import re
import random
import string
import datetime
import requests


app = Flask(__name__)
```

```
@app.route('/')
def index():
    return render_template("index.html")
```

```
@app.route("/contact")
def contact():
    return render_template("contact.html")
```

```
# conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b0aebb68-94fa-46ec-a1fc-
1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=31249;UID=mwl894
64;PASSWORD=zgelYWmImajX6HWH;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRoot
CA.crt", "", "")
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b0aebb68-94fa-46ec-a1fc-
1c999edb6187.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=31249;SECURITY=S
SL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=mwl89464;PWD=zgelYWmImajX6HW
H", '', '')
```

```
@app.route("/login", methods=['POST', 'GET'])
def loginentered():
```

```
    global Userid
    global Username
    msg = ''
    if request.method == "POST":
        email = str(request.form['email'])
        print(email)
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL=? AND PASSWORD=?"  # from db2
sql table
        stmt = ibm_db.prepare(conn, sql)
# this username & password is should be same as db-2 details & order also
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['EMAIL']
            Userid = account['EMAIL']
            session['email'] = account['EMAIL']
            Username = account['USERNAME']
            Name = account['NAME']
            msg = "logged in successfully .."
            sql = "SELECT ROLE FROM register where email = ? "
            stmt = ibm_db. prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, email)
            ibm_db.execute(stmt)
            r = ibm_db.fetch_assoc(stmt)
            print(r)
            if r['ROLE'] == 1:
                print("student")
                return render_template("studentprofile.html", msg=msg, name=Name,
role="STUDENT", username=Username, email=email)
            elif r['ROLE'] == 2:
                return render_template("facultyprofile.html", msg=msg, name=Name,
role="FACULTY", username=Username, email=email)
            else:
                return render_template("adminprofile.html", msg=msg, name=Name,
role="ADMIN", username=Username, email=email)
        else:
            msg = "Incorrect Email/password"
        return render_template("login.html", msg=msg)
    else:
        return render_template("login.html")
```

```
@app.route("/adminprofile")
def aprofile():
    return render_template("adminprofile.html")
```

```
@app.route("/adminregister", methods=['POST', 'GET'])
```

```
def signup():
    msg = ''
    if request.method == 'POST':
        name = request.form["sname"]
        email = request.form["semail"]
        username = request.form[" susername "]
        role = int(request.form[' role'])
        password = ''.join(random.choice(string.ascii_letters)
                           for i in range(0, 8))
        link = 'https://PSNAuniversity.ac.in/portal '
        print(password)
        sql = "SELECT * FROM register WHERE email=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = "Already Registered"
            return render_template('adminregister.html ', error=True, msg=msg)
```

```
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address! "
        else:
            insert_sql = "INSERT INTO register VALUES (?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, name)
            ibm_db.bind_param(prep_stmt, 2, email)
            ibm_db.bind_param(prep_stmt, 3, username)
            ibm_db.bind_param(prep_stmt, 5, role)
            ibm_db.bind_param(prep_stmt, 4, password)
            ibm_db.execute(prep_stmt)
            payload = {
                "personalizations": [
                    {
                        "to": [{"email": email}],
                        "subject": "Student Account Details"
                    }
                ],
                "from": {"email": "enter you email id"},
                "content": [
                    {
                        "type": "text/plain",
                        "value": "Dear {}, \n Welcome to PSNA university,Here
there the details to Login Into your student porta1ink :{ } YOUR Username :{} \n
PASSWORD : { } \n  Thank you \n Sincerely\n Office of Admissions\n PSNA Institute
of Technology \n E-mail: admission@PSNAuniversity.ac.in ; Website:
www.PSNAuniversity.ac.in""".format(name, link, username, password)
                    }
                ]
            }
            headers = {
```

```
                "content-type": "application/json", "X-RapidAPI-Key":
"75e4322509mshd2ae19b9a414a8cp1aa247jsn5d6e53061a20", "X-RapidAPI-Host":
"rapidprod-sendgrid-v1.p.rapidapi.com"
            }
            response = requests.request(
                "POST", url, json=payload, headers=headers)
            print(response.text)
            msg = "Registration Successful"
    return render_template('adminregister.html', msg=msg)
```

```
@app.route("/studentprofile")
def sprofile():
    return render_template('studentprofile.html')
```

```
@app.route("/studentsubmit", methods=['POST', 'GET'])
def sassignment():
    if 'Loggedin' in session:
        u = session['email']
    else:
        return redirect("/login")
    subtime = []
    ma = []
    sql = "SELECT CSUBMITTIME, MARKS from SUBMIT WHERE STUDENTNAME=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, u)
    ibm_db.execute(stmt)
    st = ibm_db.fetch_tuple(stmt)
    while st != False:
        subtime.append(st[0])
        ma.append(st[1])
        st = ibm_db.fetch_tuple(stmt)
    print(subtime)
    print(ma)
    if request.method == "POST":
        for x in range(1, 5):
            x = str(x)
            y = str("file"+x)
        print(type(y))
        f = request.files[y]
        print(f)
        print(f.filename)
        if f. filename != '':
            basepath = os.path.dirname(__file__)
            filepath = os.path.join(basepath, 'uploads', u+x+".pdf")
            f.save(filepath)
            COS_ENDPOINT = "https://control.cloud-object-
storage.cloud.ibm.com/v2/endpoints"
            COS_API_KEY_ID = "fy-22nm2nIsM6jHlBymj3GfeWiKr_DCFacfp__0mjb3H"
            COS_INSTANCE_CRN = "crn:v1:bluemix:public:cloud-object-
storage:global:a/d81536bfa337433d8b90a26200165ac2:def585e2-717e-4ed7-b79e-
3e43d4d48bc1::"
```

```
            cos = ibm_boto3.client("s3", ibm_api_key_id=COS_API_KEY_ID,
ibm_service_instance_id=COS_INSTANCE_CRN, config=Config(
                signature_version="oauth"), endpoint_url=COS_ENDPOINT)
            cos.upload_file(Filename=filepath,
                            Bucket='psnastudentassignment', key=u+x+".pdf")
            msg = "Uploding Successful"
            ts = datetime.datetime.now()
            t = ts.strftime("%Y-%m-%d %H:%M:%S")
            sql1 = "SELECT * FROM SUBMIT WHERE STUDENTNAME=? AND ASSIGNMENTNUM=?"
            stmt = ibm_db.prepare(conn, sql1)
            ibm_db.bind_param(stmt, 1, u)
            ibm_db.bind_param(stmt, 2, x)
            ibm_db.execute(stmt)
            acc = ibm_db.fetch_assoc(stmt)
            print(acc)
            if acc == False:
                sql = "INSERT into SUBMIT (STUDENTNAME,ASSIGNMENTNUM,CSUBMITTIME)
values (?,?,?)"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, u)
                ibm_db.bind_param(stmt, 2, x)
                ibm_db.bind_param(stmt, 3, t)
                ibm_db.execute(stmt)
            else:
                sql = "UPDATE SUBMIT SET CSUBMITTIME = ? WHERE STUDENTNAME =? and
ASSIGNMENTNUM=?"
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, t)
                ibm_db.bind_param(stmt, 2, u)
                ibm_db.bind_param(stmt, 3, x)
                ibm_db.execute(stmt)
            print(msg)
            return render_template("studentsubmit.html", msg=msg, subtime="2023-
05-09 12:00:00", marks=ma)
```

```
    return render_template("studentsubmit.html", subtime="2023-05-09 12:00:00",
Marks=ma)
```

```
@app.route("/facultyprofile")
def fprofile():
    return render_template("facultyprofile.html")
```

```
@app. route("/facultymarks")
def facultymarks():
    data = []
    sql = "SELECT USERNAME from REGISTER WHERE ROLE=1"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    name = ibm_db.fetch_tuple(stmt)
    while name != False:
        data . append(name)
```

```
        name = ibm_db.fetch_tuple(stmt)
    datal = []
    for i in range(0, len(data)):
        y = data[i][0].strip()
        datal.append(y)
    data1 = set(datal)
    datal = list(datal)
    print(datal)
    return render_template("facultystulist.html", names=datal, Le=len(datal))
```

```
@app.route("/marksassign/<string:stdname>", methods=['POST', 'GET'])
def marksassign(stdname):
    global u
    global g
    global file
    da = []
    COS_ENDPOINT = "https://control.cloud-object-
storage.cloud.ibm.com/v2/endpoints"
    COS_API_KEY_ID = "fy-22nm2nIsM6jHlBymj3GfeWiKr_DCFacfp__0mjb3H"
    COS_INSTANCE_CRN = "crn:v1:bluemix:public:cloud-object-
storage:global:a/d81536bfa337433d8b90a26200165ac2:def585e2-717e-4ed7-b79e-
3e43d4d48bc1::"
    cos = ibm_boto3. client(" s3", ibm_api_key_id=COS_API_KEY_ID,
ibm_service_instance_id=COS_INSTANCE_CRN, config=Config(
        signature_version="oauth"), endpoint_url=COS_ENDPOINT)
    output = cos.list_objects(Bucket="psnastudentassignment")
    output
    l = []
    for i in range(0, len(output['Contents'])):
        j = output['Contents'][i]['Key']
        l.append(j)
    l
    u = stdname
    print(len(u))
    print(len(l))
    n = []
    for i in range(0, len(l)):
        for j in range(0, len(u)):
            if u[j] == l[i][j]:
                n.append(l[i])
```

```
    file = set(n)
    file = list(file)
    print(file)
    print(len(file))
    g = len(file)
    sql = "SELECT CSUBMITTIME from SUBMIT WHERE STUDENTNAME=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, u)
    ibm_db.execute(stmt)
    m = ibm_db.fetch_tuple(stmt)
    while m != False:
```

```
        da.append(m[0])
        m = ibm_db.fetch_tuple(stmt)
    print(da)
    return render_template("facultymarks.html", file=file, marks=0, datetime=da)
```

```
@app.route("/marksupdate/<string:anum>", methods=['POST', 'GET'])
def marksupdate(anum):
    ma = []
    da = []
    mark = request.form['mark']
    print(mark)
    print(u)
    sql = "UPDATE SUBMIT SET MARKS = ? WHERE STUDENTNAME = ? and ASSIGNMENTNUM
=?"
    stm = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, mark)
    ibm_db.bind_param(stmt, 2, u)
    ibm_db.bind_param(stmt, 3, anum)
    ibm_db.execute(stmt)
    msg = "MARKS UPDATED"
    sql = "SELECT MARKS, CSUBMITTIME from SUBMIT WHERE STUDENTNAME=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, u)
    ibm_db.execute(stmt)
    m = ibm_db.fetch_tuple(stmt)
    while m != False:
        ma.append(m[O])
        da.append(m[l])
        m = ibm_db.fetch_tuple(stmt)
    print(ma)
    print(da)
    return render_template("facultymarks. html", msg=msg, marks=ma, g=g,
file=file, datetime=da)
```

```
@app.route("/logout")
def logout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    return render_template("logout.html")
```

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