

EMPLOYEE MANAGEMENT -FLASK PL2

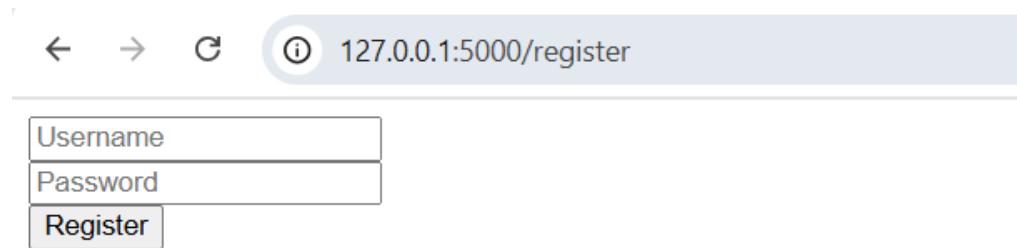
IIHT

Time To Complete: 3 hours

1 PROBLEM STATEMENT

Employee management is a Flask web application which allows users to manage employee details. The application demonstrates different HTTP methods and data input types including route variables, query strings, and JSON payloads and creating Database Base .

Register.html



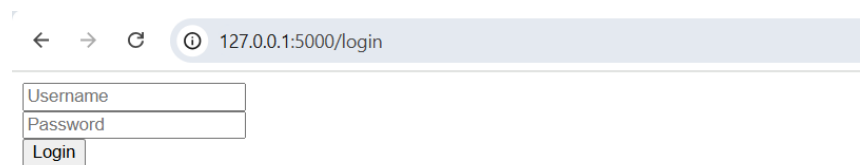
← → ↻ ⓘ 127.0.0.1:5000/register

Username

Password

Register

Login .html



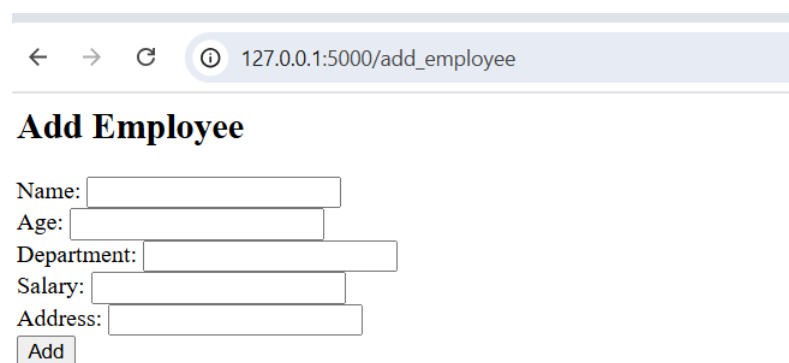
← → ↻ ⓘ 127.0.0.1:5000/login

Username

Password

Login

Add Employee



← → ↻ ⓘ 127.0.0.1:5000/add_employee

Add Employee

Name:

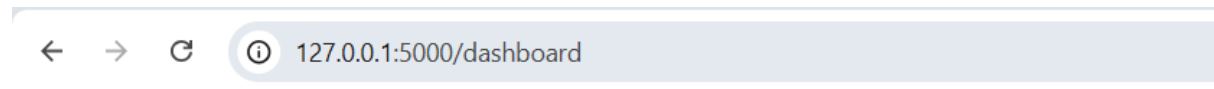
Age:

Department:

Salary:

Address:

Add



Dashboard

[Add Employee](#) | [Logout](#)

Name	Age	Department	Salary	Address	Actions
Peter Smith	35	Sales	50000	Main road, London	Edit
Alice Johnson	29	Engineering	75000	123 Elm St, New York, NY	Edit
Catherine Lee	41	Finance	85000	789 Pine Rd, Chicago, IL	Edit
Raj Patel	38	Marketing	62000	67 Queen St, Toronto	Edit
Sara Ahmed	30	HR	56000	44 King Ave, Dubai	Edit

User Story	User Story Name	User Story
Us_01	Employee management application	As a user, you should be able to add the products edit the employee details update the data and create require data in the database .

The SQLite sample data is already given to the users you can import the data by running the command `python3 db_init.py`. After you have executed the command, you can see the sample data loaded in the database.

Name	Age	Department	Salary	Address
Peter Smith	35	Sales	50000	Main road, London
Alice Johnson	29	Engineering	75000	123 Elm St, New York, NY
Catherine Lee	41	Finance	85000	789 Pine Rd, Chicago, IL
Raj Patel	38	Marketing	62000	67 Queen St, Toronto
Sara Ahmed	30	HR	56000	44 King Ave, Dubai

#	Functionality	Route	Method(s)	Details
1	Display homepage	/	GET	Renders <code>index.html</code> . Entry point of the application.
2	User registration	/register	GET, POST	GET: Show <code>register.html</code> form. POST: Saves user (username, password) to users table in SQLite DB.
3	User login	/login	GET, POST	GET: Show <code>login.html</code> . POST: Verifies credentials, starts session, redirects to /dashboard.
4	Display user dashboard	/dashboard	GET	Displays <code>dashboard.html</code> with employee list (requires login).
5	Add employee (form-based)	/add_employee	GET, POST	GET: Show <code>add_employee.html</code> . POST: Adds new employee to employees table. Requires active session.
6	Edit employee by ID	/edit_employee/<int:id>	GET, POST	GET: Shows form with pre-filled employee data. POST: Updates employee record in the database.
7	Add employee (API)	/api/employee	POST	Accepts JSON payload with name, salary, age, address. Adds new employee to database. Returns JSON response.
8	Fetch all employees (API)	/employees	GET	Returns all employee records as JSON. Handles internal errors gracefully.
9	Logout user	/logout	GET	Ends user session by removing <code>user_id</code> from session. Redirects to home page.

Resources views

Template File	Description	Status
<code>index.html</code>	The Home page for the eCommerce application	Already implemented
<code>register.html</code>	User registration page for creating a new account	Already implemented

Template File	Description	Status
login.html	Login page for existing users to authenticate	Already implemented
dashboard.html	Dashboard displaying the employee list (post-login)	Already implemented
add_employee.html	Form to add a new employee record	Already implemented
edit_employee.html	Form to update existing employee details	Already implemented

In App.py

Functions to be implemented

def get_db_connection():	<ul style="list-style-type: none"> The user needs to create the database for the Flask app To initialize the database you will to run the init_db.py Create a connection to employees.db Return the connection object 	To be implemented
def index():	<ul style="list-style-type: none"> This renders the index.html The user needs the def function to map to index.html The template should contain a welcome message and links to register/login 	To be implemented
def register():	<ul style="list-style-type: none"> Get username and password from the form. The default username and password is 'admin' and 'admin 123' Insert new user into the users table Redirect to the login page after successful registration 	To be implemented
def login():	<ul style="list-style-type: none"> Get the username and password from the form Check credentials against the users table If valid, store user_id in session and redirect to the dashboard If invalid, return error message, If user found, store user_id in session and redirect to dashboard If the user is not found, return 'Invalid credentials' 	To be implemented

def dashboard():	<ul style="list-style-type: none"> • Check if user is logged in (user_id in session) • - If not logged in, redirect to login • - If logged in, get all employees from the database and render the dashboard • All the employees should be visible on the dashboard. 	To be implemented
def add_employee():	<ul style="list-style-type: none"> • The add employee should be mapped to add_employee.html • Get name, salary, age, address from form , • Connect to the database • Insert employee data into the employees table • Commit and close the connection • Redirect to dashboard 	To be implemented
def edit_employee(id):	<ul style="list-style-type: none"> • Get updated employee data from the form; the data to be updated is mentioned below. • Update employee record in the database • Commit and close the connection • Redirect to dashboard 	To be implemented
def add_employee_api(id):	<ul style="list-style-type: none"> • The data to be added through JSON is mentioned below: Extract name, salary, age, and address from JSON • Connect to the database • Insert employee data • Commit and close the connection 	To be implemented
def get_all_employee():	<ul style="list-style-type: none"> • Use this URL /employees using GET to display all the employees from the data through Postman • Format results as a list of dictionaries 	To be implemented
def logout():	<ul style="list-style-type: none"> • Remove user_id from the session • Redirect to the index page 	To be implemented

1)The Json Data to be used for post man add_employee_api ()

```
{
'name': 'Alice Watson',
'salary': '72000',
'address': '55 Sunset Blvd, LA'
}
```

2)The Employee details to be edited for Raj Patel from 38 to 58 in the frontend edit_employee(id)

3) The Employee details to be added employee is Jhon', '56000', '24 King Ave, Dubai add_employee():

The USER IS required to use the POST MAN client to use the GET AND POST requests .

2 MANDATORY ASSESSMENT GUIDELINES

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. The editor Auto Saves the code.
4. If you want to exit(logout) and to continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
7. To test any UI based application the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.
8. Install flask module before running the code. For this use the following command.
`pip install flask`
9. Use the following command to
run the server `python3 app.py`
10. Mandatory: Before final submission run the following commands to
execute testcases `python3 -m unittest`

11. To test rest end points

Click on 'Thunder Client' or use Ctrl+Shift+R ->Click on 'New Request' (at left side of IDE)

12. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.
13. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.