Python Programming Test - LeanKloud

	PART 1
1.	Flask-restplus Implementation for REST API
2.	Implement the storage for tasks and database
3.	Add 2 new fields to a task : Due by,Status
4.	Add or modify web methods to change the status of tasks.
5.	Implement the end points - "GET /due?due_date=yyyy-mm-dd"
6.	Implement the end points - "GET /overdue"
7.	Implement the end points - ""GET /finished"
8.	Implement authorization - To secure endpoints
9.	Role based web access
	PART 2
10.	Find the topper in each subject.
11.	Top 3 students in the class, based on their marks in all subjects.
12.	Print result on console with csv as argument
13.	Complexity

Dependencies:

Python version: 3.7

```
pip3 install Werkzeug==2.0.3
pip3 install Flask==2.1.2
pip3 install flask flask-restplus==0.13.0
```

To run: cd into the app.py directory and python3 app.py

PART 1

1. Flask-rest Plus Implementation for REST API:

Flask-RESTPlus is an extension for Flask that adds support for quickly building REST APIs. Flask-RESTPlus encourages best practices with minimal setup.

1. The REST API calls implemented in this project:

To get all tasks:
http://localhost:5000/tasks/ GET

To add new tasks:
http://localhost:5000/tasks/ POST

To update an existing task:
http://localhost:5000/tasks/id PUT

To delete an existing task:
http://localhost:5000/tasks/id DELETE

To get overdue tasks:
http://localhost:5000/tasks/overdue GET

To get finished tasks:
http://localhost:5000/tasks/finished GET

To get all task based on date
http://localhost:5000/tasks/due?due_date=yyyy-mm-dd GET

If to test the API in postman then for the POST and PUT method's API the payload needs to be passed in body.

```
{
   "task": "python",
   "due_date": "2023-10-19",
   "status": "in progress" ( provide one of these: Not started, In progress, Finished")
}
```

And Authorization in the header as key and token as value http://127.0.0.1:5000/register: To check this API in postman the payload will be like {"username": "ashok","password": "3", "user_role": "0"} -When Write Access is selected then 1 (admin) get passed for read access 0 passed (from drop-down)

http://127.0.0.1:5000/login:To check this API in postman the payload will be like {"username": "ashok", "password": "3"} - the data which we gave in register and the response like we get auth token, use that token to the above Flask REST Plus API in the header. Each time the login API is called it generates a different auth token

http://127.0.0.1:5000/logout: No need to pass any payload or header, it logs out from the browser

2. The routing calls implemented in this project:

http://127.0.0.1:5000/register:
The user can create an entry for the user (any number of times) based on username & can't re-register with the same username again
(username, password, user role)

http://127.0.0.1:5000/login:
To log in to the web page use the registered user credentials - username and password.

http://127.0.0.1:5000/get_all_todos:
On this page, we get all the created tasks.

http://127.0.0.1:5000/addtodo:
On this page, we create a new task.

http://127.0.0.1:5000/overdue:
On this page, we get all the overdue created tasks.

http://127.0.0.1:5000/finished:
On this page, we get all the finished tasks.

http://127.0.0.1:5000/due?due_date=YYYY-MM-DD:
On this page, we get task details based on the date which is passed in the link as a query parameter.

http://127.0.0.1:5000/edit_todo/id:
On this page, we edit the existing task

2. Implement the storage for tasks and database:

To store the data, I have created two databases for the storage purpose
todo.db - it has the table “tasks” where all the task-related data get stored
users.db - it has the table “users” where all the user's credentials get stored while registering. The database used in this project is **SQLite** is a lightweight db.

3. Add 2 new fields to a task: Due by, Status & 4. Add or modify web methods to change the status of tasks

This project contains three fields where we can add, edit, and delete those fields, the fields are **TASK, DUE_BY, and STATUS** by using the above-mentioned REST-Plus API calls a user can access all of the these data.

5. Implement the endpoints - "GET /due?due_date=yyyy-mm-dd" , 6. Implement the endpoints - "GET /overdue" & 7. Implement the endpoints - "GET /finished"

As mentioned earlier, these API endpoints retrieve related data from the database. Users can view this data on the UI through the specified routing URLs.

8. Implement authorization - To secure endpoints:

RESTPlus APIs are accessed through specific endpoints, which can be secured by restricting access to authorized users only. To achieve this, each endpoint requires an **authorization token** passed in the **header**. The API validates this token, ensuring that the request is valid and authorized. This mechanism prevents unauthorized access to API calls associated with specific routing URLs, enforcing the requirement for users to log in to the website before accessing these endpoints.

9. Role-based web access:

In a role-based user system, individuals with write access possess the ability to create, modify, and delete task details. On the other hand, users with read-only access can solely view the data and are restricted from performing any operations on the UI. This role-based approach ensures that users are granted appropriate privileges based on their level of access, maintaining security and control over the system's functionalities.

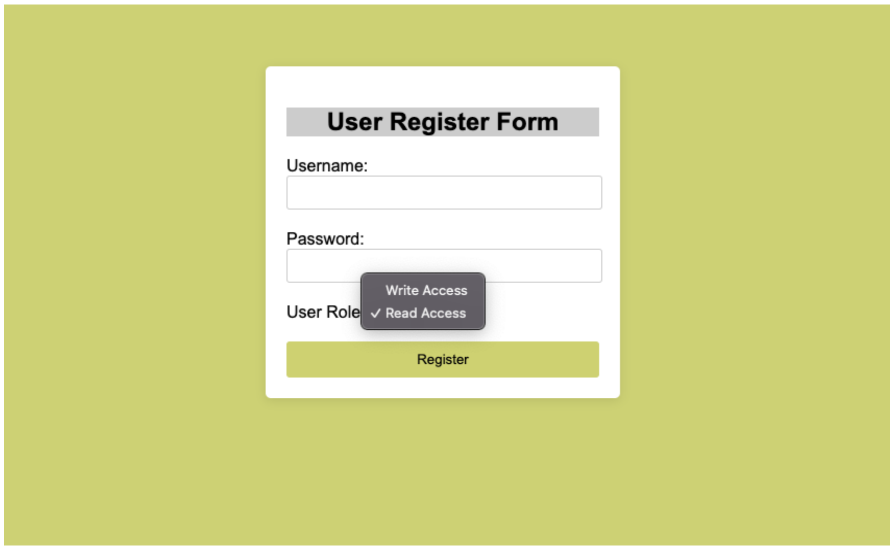
For demo purposes:

I have registered two users one with a write access user and a read access. You can register another user on the register page and use those credentials to log in

Write Access
username - gowtham
password - 1

Read Access
username - sriram
password - 2

fig:1
http://127.0.0.1:5000/register:
When Write Access is selected then 1 (admin) get passed for read access 0 passed


User Register Form

Username:

Password:

User Role: [✓ Read Access]

fig:2

http://127.0.0.1:5000/login:

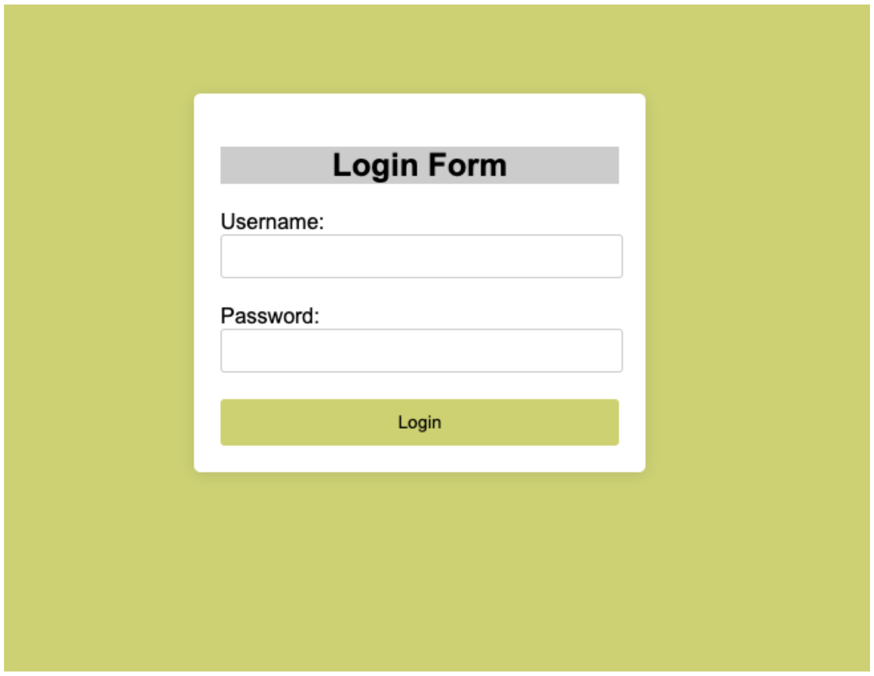
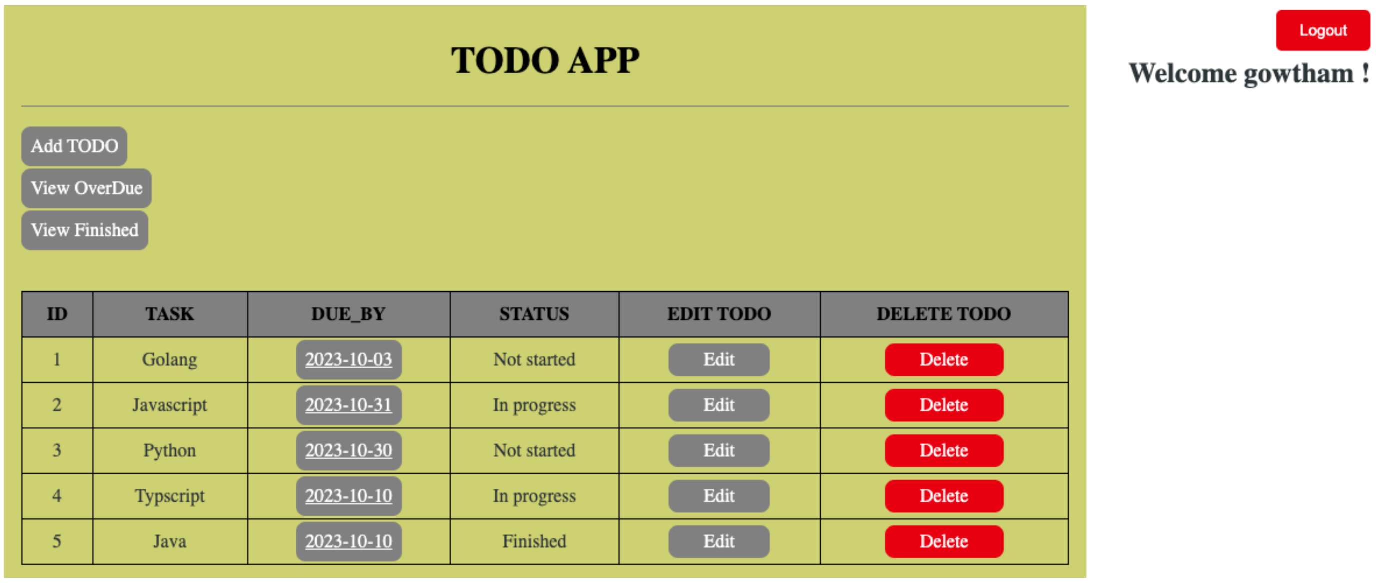


fig:3

http://127.0.0.1:5000/get_all_todos:

WRITE ACCESS UI

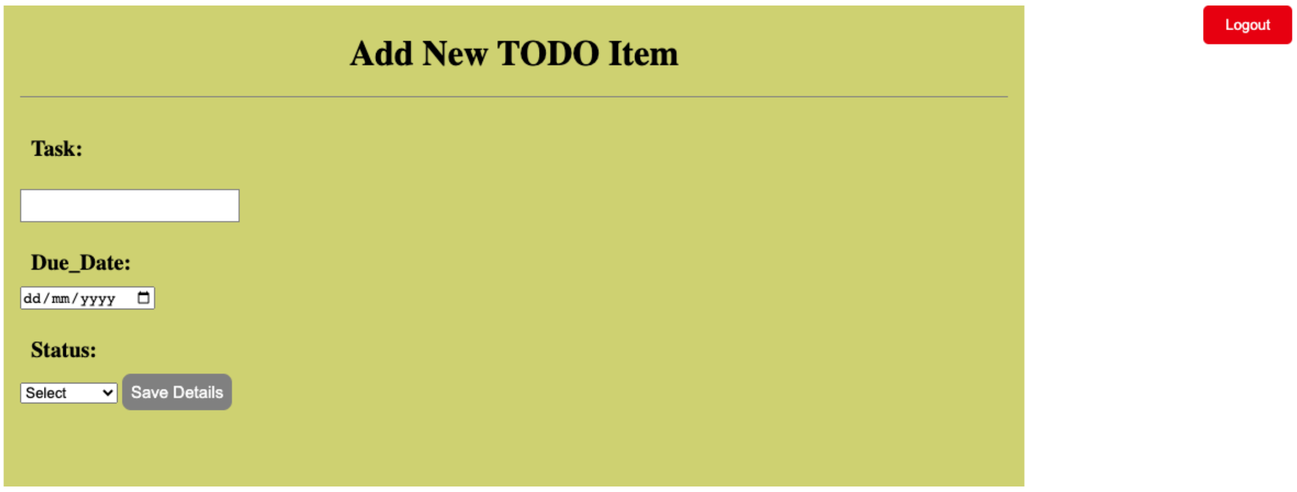


ID	TASK	DUE_BY	STATUS	EDIT TODO	DELETE TODO
1	Golang	2023-10-03	Not started	Edit	Delete
2	Javascript	2023-10-31	In progress	Edit	Delete
3	Python	2023-10-30	Not started	Edit	Delete
4	Typscript	2023-10-10	In progress	Edit	Delete
5	Java	2023-10-10	Finished	Edit	Delete

fig:4

http://127.0.0.1:5000/addtodo:

By clicking the Add TODO button in the previous fig:3



Logout

Add New TODO Item

Task:

Due_Date:

dd/mm/yyyy

Status:

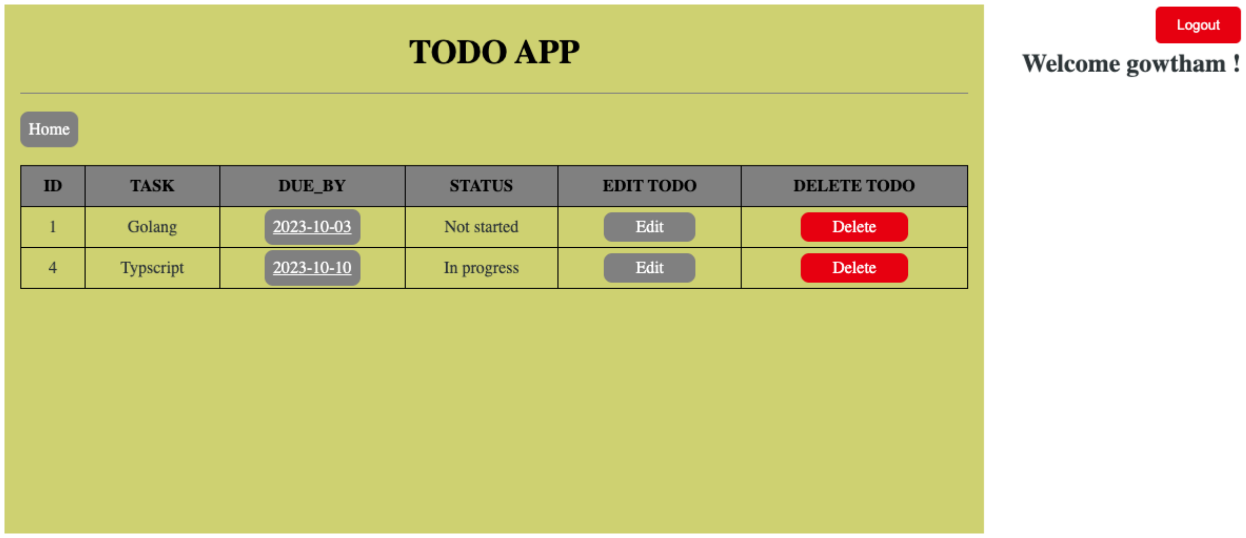
Select

Save Details

fig:5

http://127.0.0.1:5000/overdue:

By clicking the View OverDue button in the previous fig:3



Logout

Welcome gowtham !

TODO APP

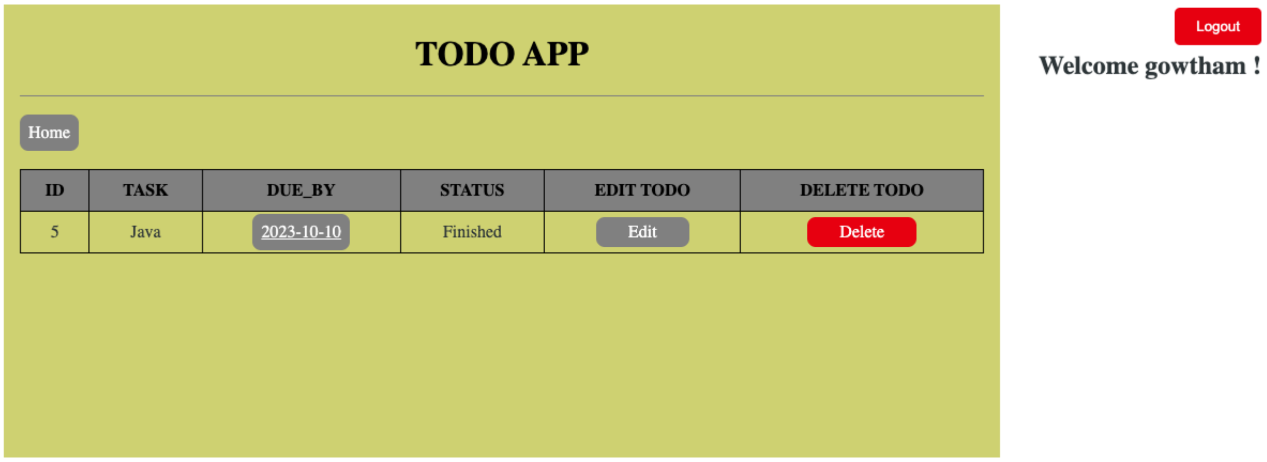
Home

ID	TASK	DUE_BY	STATUS	EDIT TODO	DELETE TODO
1	Golang	2023-10-03	Not started	Edit	Delete
4	Typscript	2023-10-10	In progress	Edit	Delete

fig:6

http://127.0.0.1:5000/finished:

By clicking the View Finished button in the previous fig:3



Logout

Welcome gowtham !

TODO APP

Home

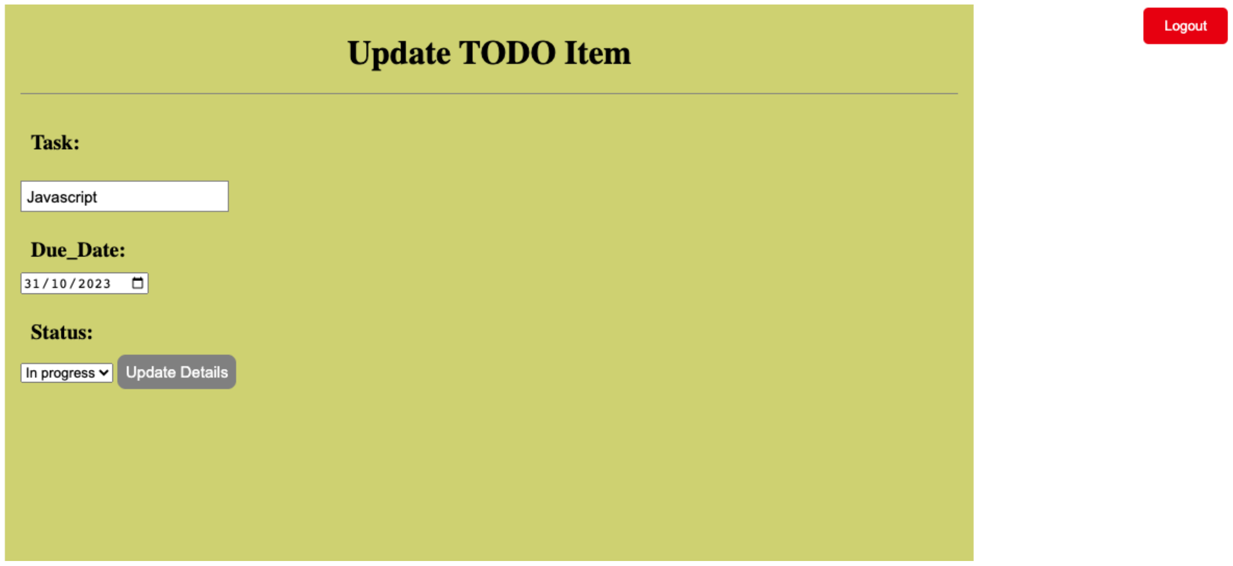
ID	TASK	DUE_BY	STATUS	EDIT TODO	DELETE TODO
5	Java	2023-10-10	Finished	Edit	Delete

Fig:7

http://127.0.0.1:5000/edit_todo/{id}:

The value id in the URL is the id against the edit you clicked

By clicking the Edit button



Logout

Update TODO Item

Task:

Javascript

Due_Date:

31/10/2023

Status:

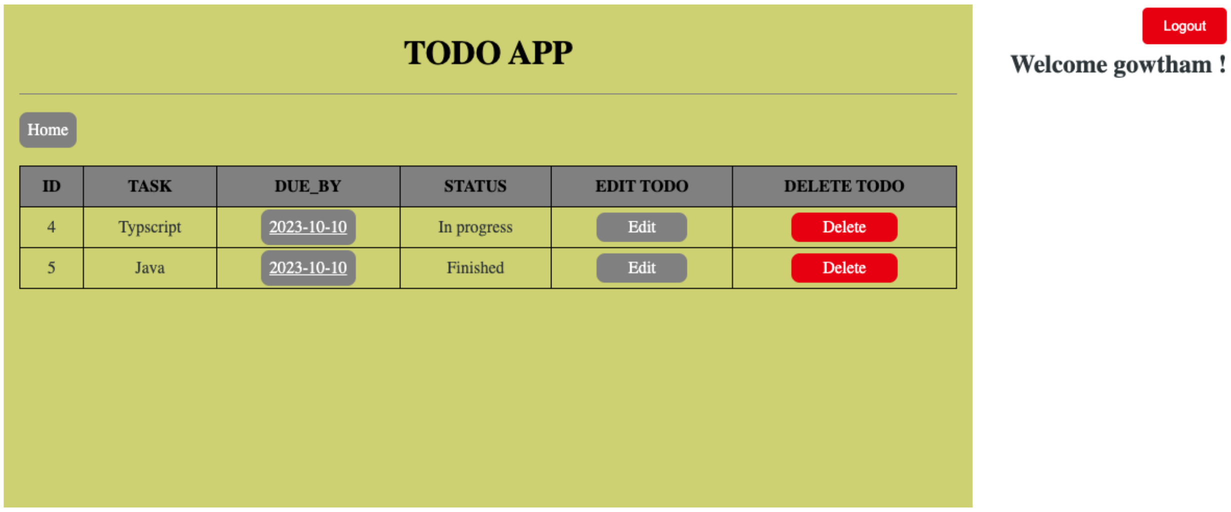
In progress

Update Details

Fig:8

http://127.0.0.1:5000/due?due_date=YYYY-MM-DD:

There is a same date with two entry in the fig:3, so if we click on the date in the **Due_By** column then it will show the tasks belongs to that particular date



Welcome gowtham !

TODO APP

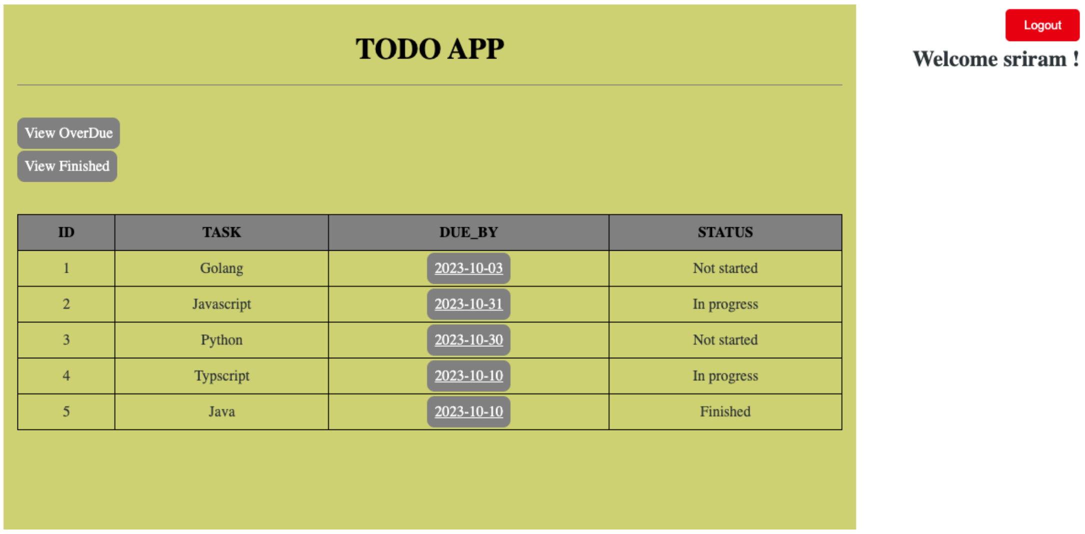
ID	TASK	DUE_BY	STATUS	EDIT TODO	DELETE TODO
4	Typscript	2023-10-10	In progress	Edit	Delete
5	Java	2023-10-10	Finished	Edit	Delete

Fig:9

http://127.0.0.1:5000/get_all_todos:

READ ACCESS UI

In this page user can only view the data by clicking View OverDue, View Finished & clicking date in the Due_BY column.



Welcome sriram !

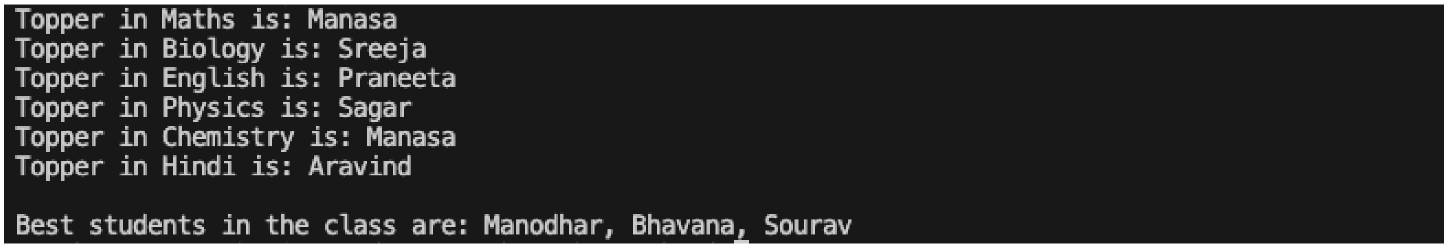
TODO APP

ID	TASK	DUE_BY	STATUS
1	Golang	2023-10-03	Not started
2	Javascript	2023-10-31	In progress
3	Python	2023-10-30	Not started
4	Typscript	2023-10-10	In progress
5	Java	2023-10-10	Finished

PART 2

10. Find the topper in each subject,
11. Top 3 students in the class, based on their marks in all subjects,
12. print result on console with csv as argument.

This Python script reads student marks data from a CSV file, identifies subject-wise toppers, and lists the top three students based on total marks. It checks for the file's existence, processes the data, and displays subject-wise toppers and overall best students. This algorithm builds without using Panda.



```
Topper in Maths is: Manasa
Topper in Biology is: Sreeja
Topper in English is: Praneeta
Topper in Physics is: Sagar
Topper in Chemistry is: Manasa
Topper in Hindi is: Aravind

Best students in the class are: Manodhar, Bhavana, Sourav
```

13.Complexity

- read_csv(): O(n)
- find_topper(): O(n log n)
- find_top_students(): O(n log n)
- print_results(): O(n)

The overall complexity of the program is O(n log n), since the two most expensive operations, find_topper() and find_top_students(), are both O(n log n). Therefore, the **complexity** of the program is **O(n log n)**.