UniTask

Bases de Dados
Secondary Report

Francisco Cardita 98501

Pedro Ferreira 102876

2022/2023

Table of Contents

Introduction

The aim of this platform is to build a web application that helps manage tasks for university students.

Technologies used

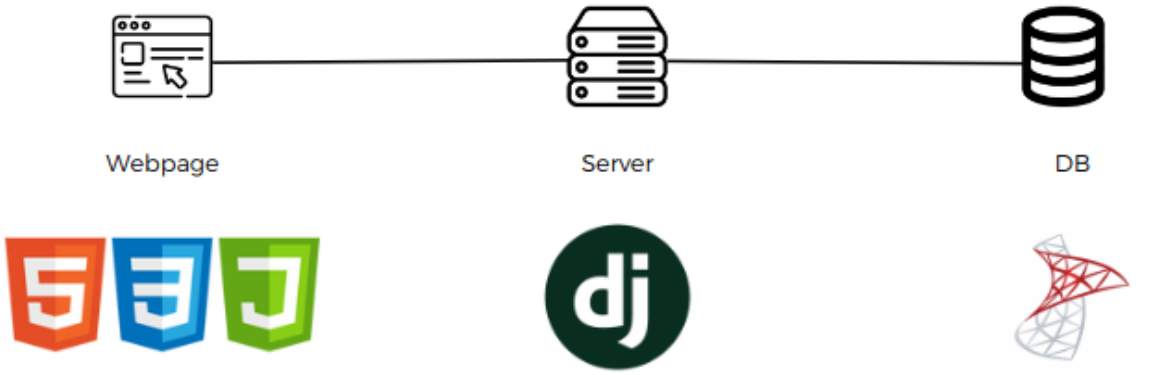
Frontend

- HTML
- CSS (Bootstrap)
- JavaScript (jQuery)

Backend

- Django
- Microsoft SQL Server

Infrastructure



Requirements

To run the server, you need to have the following software installed:

- Docker
- Docker compose

Getting Started

---

1. Clone the repository:

```
git clone <repo-url>
```

2. Navigate to the project directory.

Build and run the Docker containers:

```
docker compose up
```

3. Now wait for 60 seconds for the containers to start up, and the DB to be seeded with data.

4. Once everything is up and running, you can access the web platform in your browser at **http://localhost:8000/**.

Layout

The project folder should look like this:

```
.
├── db/
└── uni_todo/
  ├── api/
  ├── app/
  └── uni_todo/
```

- `db/`: Contains scripts and an sql file to run at startup;
- `uni_todo/`: Django project;
  - `api/`: Handles api requests' logic;
  - `app/`: Serves html pages and static files;

API Endpoints

---

1. **`/list_universities`**
   - Request method: GET
   - Description: Retrieves a list of universities from the database.
   - Example response:

```
[
 {
   "id": 1,
   "name": "University of Aveiro"
 },
 {
   "id": 2,
   "name": "Cambridge University"
```

```
  }
]
```

2. **`/delete_task`**
    - Request method: POST
    - Description: Soft deletes a task from the database.
    - Example request:

```
{
  "task_id": "123"
}
```

    - Example response:

```
{
  "message": "Task deleted successfully."
}
```

3. **`/associate_task_with_user`**
    - Request method: POST
    - Description: Associates a task with a user.
    - Example request:

```
{
  "task_id": "123",
  "user_id": "456"
}
```

- Example response:

```
 {
  "message": "Task associated with user successfully."
}
```

4. **`/follow_user`**
    - Request method: POST
    - Description: Follows a user.
    - Example request:

```
{
  "follower_id": "123",
  "followee_id": "456"
}
```

    - Example response:

```
 {
  "message": "User followed successfully."
}
```

5. **`/unfollow_user`**
    - Request method: POST
    - Description: Unfollows a user.
    - Example request:

```
 {
  "follower_id": "123",
  "followee_id": "456"
}
```

    - Example response:

```
{
  "message": "User unfollowed successfully."
}
```

6. **`/list_followees`**
    - Request method: GET
    - Description: Retrieves a list of followees for a given user.
    - Example request: `/list_followees?user_id=123`
    - Example response:

```
[
  {
    "id": 1,
    "name": "User 1",
    "uni_id": 123
```

```
  },
  {
    "id": 2,
    "name": "User 2",
    "uni_id": 456
  }
]
```

7. **`/list_followers`**
    - Request method: GET
    - Description: Retrieves a list of followers for a given user.
    - Example request: `/list_followers?user_id=123`
    - Example response:

```
[
  {
    "id": 1,
    "name": "Follower 1",
    "uni_id": 123
  },
  {
    "id": 2,
    "name": "Follower 2",
    "uni_id": 456
  }
]
```

8. **`/list_tasks`**
    - Request method: GET
    - Description: Retrieves a list of tasks for a given user.
    - Example request: `/list_tasks?user_id=1&is_public=0`
    - Example response:


```
[
  {
   "task_name": "New Task",
   "class_id": 123,
   "description": "Task description",
   "group": "Group 1",
   "status": "Pending",
   "start_date": "2023-05-28",
   "end_date": "2023-06-05",
   "priority_lvl": 2
 },
 {
   "task_name": "New Task 2",
   "class_id": 1,
   "description": "Task description 2",
   "group": "Group 2",
   "status": "Completed",
   "start_date": "2023-05-28",
   "end_date": "2023-06-05",
   "priority_lvl": 5
 }
]
```

9. `/create_task`
    - Request method: POST
    - Description: Creates a new task.
    - Example request:

```
 {
  "task_name": "New Task",
  "class_id": 123,
  "description": "Task description",
  "group": "Group 1",
  "status": "Pending",
  "start_date": "2023-05-28",
  "end_date": "2023-06-05",
  "priority_lvl": 2,
  "is_public": 1,
  "user_id": 456
}
```

    - Example response:

```
{
  "message": "Task created successfully."
}
```

10. `/search_user`
    - Request method: POST
    - Description: Searches for a user by name and returns users with a flag indicating whether the current user can follow them.

- Example request:

```
 {
  "user_name": "John",
  "user_id": "123"
}
```

- Example response:

```
 [
  {
    "id": 1,
    "name": "John",
    "uni_id": 123,
    "can_follow": true
  },
  {
    "id": 2,
    "name": "John Doe",
    "uni_id": 456,
    "can_follow": false
  }
]
```

11. **`/get_user`**
    - Request method: GET
    - Description: Retrieves a user and university name by user id.
    - Example request: `/get_user?user_id=123`
    - Example response:

```
{
  "id": 1,
  "name": "John",
  "university": "University of Aveiro"
}
```

12. **`/update_task`**
    - Request method: POST
    - Description: Updates a task.
    - Example request:

```
{
  "task_id": "123",
  "task_name": "Updated Task",
  "class_id": "456",
  "description": "Updated task description",
  "group": "Updated Group",
  "status": "In Progress",
  "start_date": "2023-06-01",
  "end_date": "2023-06-10",
  "priority_lvl": 1,
  "is_public": 0
}
```

- Example response:

```
 {
  "message": "Task updated successfully."
}
```

13. **`/register_user`**
    - Request method: POST
    - Description: Registers a new user.
    - Example request:

```
{
  "username": "newuser",
  "password": "password123"
}
```

    - Example response:

```
 {
  "user_id": 123
}
```

This endpoint uses the Django library django.db.transaction.atomic(), to ensure both queries are successfully executed, before committing to DB.

14. **`/login_user`**
    - Request method: POST

- Description: Logs in a user.
- Example request:

```
 {
  "username": "user",
  "password": "password"
}
```

- Example response:

```
 {
  "status": true,
}
```

15. **`/list_classes`**
    - Request method: GET
    - Description: Retrieves a list of classes for the university the user attends.
    - Example request: `/list_classes?user_id=123`
    - Example response:

```
 [
  {
    "id": 1,
    "name": "Class 1"
  },
  {
    "id": 2,
    "name": "Class 2"
  }
]
```

Note:

The logic for all these endpoints can be found at `uni_todo/api/views.py`, with each function corresponding to the endpoint url name. Those functions execute stored procedures in DB, for more information about them, please refer to `report.pdf`.