API documentation

COMP2000 Coursework API Documentation

The API allows students to perform CRUD operations on their personal databases. The API is accessible via:

```
http://10.240.72.69/comp2000/coursework/
```

This documentation includes the available endpoints, their expected inputs, and the responses.

---

1. Create Student Database

Endpoint: `/create_student/{student_id}`

Method: `POST`

Input Data:

- **Path Parameter**:
  - `student_id`: Unique identifier for the student (e.g., `student_123`)

Response:

- **Status Code**: 200 (OK)
- **Body**:

  ```
  {
    "message": "Student database created successfully"
  }
  ```

---

2. Create User Entry

Endpoint: `/create_user/{student_id}`

Method: `POST`

Input Data:

- **Path Parameter**:
    - `student_id`: The unique identifier of the student (e.g., `student_123`)
- **Request Body**:
    - `username`: The username of the user
    - `password`: The password of the user
    - `firstname`: The user's first name
    - `lastname`: The user's last name
    - `email`: The user's email address
    - `contact`: The user's contact number
    - `usertype`: The type of user (e.g., "student", "admin")

  Example input JSON:

  ```
  {
    "username": "john_doe",
    "password": "test",
    "firstname": "John",
    "lastname": "Doe",
    "email": "john.doe@example.com",
    "contact": "1234567890",
    "usertype": "student"
  }
  ```

Response:

- **Status Code**: 200 (OK)
- **Body**:

```
{
  "message": "User created successfully"
}
```

---

3. Read All Users

Endpoint: `/read_all_users/{student_id}`

Method: `GET`

Input Data:

- **Path Parameter**:
  - `student_id`: The unique identifier of the student (e.g., `student_123`)

Response:

- **Status Code**: 200 (OK)
- **Body**:

```
{
  "users": [
    {
      "username": "john_doe",
      "password": "test",
      "firstname": "John",
      "lastname": "Doe",
      "email": "john.doe@example.com",
      "contact": "1234567890",
      "usertype": "student"
    },
    {
      "username": "jane_doe",
      "password": "test",
      "firstname": "Jane",
      "lastname": "Doe",
      "email": "jane.doe@example.com",
      "contact": "9876543210",
      "usertype": "staff"
    }
  ]
}
```

---

4. Read Specific User

Endpoint: `/read_user/{student_id}/{username}`

Method: `GET`

Input Data:

- **Path Parameters**:
    - `student_id`: The unique identifier of the student (e.g., `student_123`)
    - `user_id`: The unique identifier of the user (e.g., `user_456`)

Response:

- **Status Code**: 200 (OK)
- **Body**:

```
{
  "user": {
    "username": "john_doe",
    "password": "test",
    "firstname": "John",
    "lastname": "Doe",
    "email": "john.doe@example.com",
    "contact": "1234567890",
    "usertype": "student"
  }
}
```

---

5. Update User

Endpoint: `/update_user/{student_id}/{username}`

Method: `PUT`

Input Data:

- **Path Parameters**:
  - `student_id`: The unique identifier of the student (e.g., `student_123`)
  - `user_id`: The unique identifier of the user (e.g., `user_456`)
- **Request Body**:
  - `username`: The updated username of the user
  - `password`: The password of the user
  - `firstname`: The updated first name
  - `lastname`: The updated last name
  - `email`: The updated email address
  - `contact`: The updated contact number
  - `usertype`: The updated user type

  Example input JSON:

```
{
  "username": "john_doe_updated",
  "password": "test",
  "firstname": "John",
  "lastname": "Doe",
  "email": "john.doe_updated@example.com",
  "contact": "9876543210",
  "usertype": "student"
}
```

Response:

- **Status Code**: 200 (OK)
- **Body**:

```
{
  "message": "User updated successfully",
  }
```

---

6. Delete User

Endpoint: `/delete_user/{student_id}/{username}`

Method: `DELETE`

Input Data:

- **Path Parameters**:
  - `student_id`: The unique identifier of the student (e.g., `student_123`)
  - `user_id`: The unique identifier of the user (e.g., `user_456`)

Response:

- **Status Code**: 200 (OK)
- **Body**:

```
{
  "message": "User deleted successfully"
}
```

Error Handling

- **404 - Not Found**: If the requested user is not found in the database.

```
{
  "detail": "User not found"
}
```

- **400 - Bad Request**: If the input data is invalid (e.g., missing required fields).

  ```
  {
    "detail": "Invalid user data"
  }
  ```
