There are three primary datatables users can access from the API. The first is the events table. It has the following fields for a given event(user accessible fields are bolded):

1. id(integer, automatically incremented)
2. **hostName(required String)**
   a. This is essentially the club hosting an event
3. **title(required String)**
   a. This is essentially the name of the event
4. **description(required String)**
5. **time(required String)**
   a. Of the format "dd/mm/yy hr:min"
6. **location(required String)**
7. Students
   a. This is the students signed up for an event in array form

The API will start off with 4 usable events already in the events datatable. The second table is the students table. It has the following fields for a given student(user accessible fields are bolded):

1. id(integer, automatically incremented)
2. **name(required String)**
3. **netid(required String)**
4. **majorrequired String)**
5. **grade(required String)**
6. Events
   a. This is the events the student is signed up for an event in array form
7. User
   a. This is private information about the user such as their email and password.

The third table is the User datable that handles authentication. It is associated with the Student table using foreign keys, and the code cheks to make sure that only one email and password are connected to a student. The User table has the following fields:

1. id(integer, automatically incremented)
2. **email(required String, must be unique)**
3. **password(required String)**
4. session_token(required String, must be unique)
5. session_expiration(required Datetime object)
6. update_token(required String, must be unique)
7. student_id(integer, associates to Student table)

The "homepage" of the API can be accessed at 35.225.157.221
In addition to the two datatables, the API has 5 main routes:

1. Get all events
    a. URL: (/api/events/)
2. Create an event
    a. URL: (/api/events/)
    b. Example Request Format:

```
{
  "hostName": "Cornell",
  "title": "Slope Day",
  "description": "Best day of the year",
  "time": "10/23/2022 15:30",
  "location": “Slope"
}
```

3. Get an event by a specific date and time
    a. URL: (/api/events/<string:time>/)
    b. Gets the first event for that specific date and time
4. Create a student
    a. URL: (/api/students/)
    b. Example Request Format:

```
{
  "name": "Blake",
  "netid": "r3522",
  "major": "Compouter Science",
  "grade": "Sophomore"
}
```

5. Delete student by name
    a. URL: (/api/student/<string:name>/)
6. Delete event by id
    a. URL: (/api/event/<int:id>/)

The API also has 5 routes related to handling authentication.

1. Register a password and email for a user
    a. URL: (/register/)
    b. Example request Format:

```
{
   "email":"rg536@cornell.edu",
    "password":"This is my password",
    "studentid": 3
}
```

2. Logs in a user given the correct password and email
    a. Example request Format:

```
{
    "email":"rg536@cornell.edu",
    "password":"This is my password"
}
```

3. Updates user session
    a. URL:(/session/)
    b. Update token must be provided through use of the header
4. Logs user out of session
    a. URL:(/logout/)
    b. Session token must be provided through use of the header

As a reminder, the <> in URLs should be switched with your actual value that you want to get.
Note: User was implemented as a separate table as our team decided to do authentication on the frontend.
Note: There is also an authentication route that lets users check if authentication has worked by returning a secret message, this route is located at URL: (/secret/)