

# Documentation

## Important Utils

Following packages are extra installed while implementing backend functionality:

```
$ npm install bcrypt  
$ npm install body-parser
```

## Backend Documentation

### POST /login

**Description:** Use email and password to check whether this user is valid in the database. Return a JSON object with all the user information with a “check” field equals to 1 if succeed and return a JSON object with only a “check” field equals to 0 if failed.

#### Body Parameters:

- email: String
- password: String

#### Expected Response:

Successfully authenticated:

Return

```
{  
  "check": 1,  
  "_id": "633c847a87bbbf6f3bd1361c",  
  "email": {  
    "data": "caleb@123.com",  
    "display": true  
  },  
  "password": "$2b$10$grXWK0rmF4WdriG7jVyu2OzXLwj1xdYtzJw8q/gkBQr18EY60O0ti",  
  "name": {  
    "data": "",  
    "display": true  
  },  
  "dateofbirth": {  
    "data": null,  
    "display": true  
  },  
  "gender": {  
    "data": "",  
    "display": true  
  },  
  "Program": {  
    "data": "",  
    "display": true  
  },  
}
```

```
        "display": true
    },
    "Description": {
        "data": "",
        "display": true
    }
}
```

Failed:

Return

```
{
  "check": 0
}
```

### POST /signup

**Description:** adds a new student to the student collection in the database for newly registered users, return a json file which contains attribute result.

**Body Parameters:**

- Email: The email of the new registered student.
- Password: The password provided by the new student.

**Expected Response:**

- {result:"1"}

The student has no duplicate email as another student in the database.

A screenshot of a REST client interface. At the top, there are tabs for 'Body', 'Cookies', 'Headers (9)', and 'Test Results'. On the right, there are status indicators for '200 OK' and '1342 ms'. Below these are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', 'JSON', and a copy icon. The main area shows a JSON response with line numbers 1, 2, and 3. Line 1 shows an opening brace {}, line 2 shows the key "result": "1", and line 3 shows a closing brace }.

```
1  {
2   "result": "1"
3 }
```

- {result:"0"}

The student has no duplicate email as another student in the database or there is an error during the connection process.

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

1
2   "result": "0"
3

```

## DELETE /deleteUser

**Description:** Delete an existing user with the corresponding email passed by the body parameters from the student collection in the database, return a json file which contains 3 parts including the req (all body parameters), result of 0 (failure) /1 (success), and a message.

### Body Parameters:

- Required
  - Email: The email of the existing student.
- Optional
  - All other information

### Expected Response:

- Body parameter example:

Authorization Headers (1) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
  "_id": {"$oid": "633efce10cc286be9c9f6b9f"}, "email": {"data": "alfred@utoronto.ca", "display": true}, "password": "$2b$10$beEJ3TqgTvQTxhc4/yCFj
  ./FIPjA7b1GPFl1bb7tgTVTwFeOHxu", "name": {"data": "", "display": true}, "dateOfBirth": {"data": null, "display": true}, "gender": {"data": "", "display": true}
  , "Program": {"data": "", "display": true}, "Description": {"data": "", "display": true}}

```

Note: Only the email part is needed to delete a user, but this is just an example with all the optional information

- On success (existing user with the given email is deleted in database):

```
1 ▼ {
2 ▼   "data": {
3 ▼     "_id": {
4       "$oid": "633efce10cc286be9c9f6b9f"
5     },
6 ▼     "email": {
7       "data": "alfred@utoronto.ca",
8       "display": true
9     },
10    "password": "$2b$10$beEJ3TqgTvQTxhc4/yCFj./FFIPjA7b1GPF11bb7tgTVTwFeOHvXu",
11    "name": {
12      "data": "",
13      "display": true
14    },
15    "dateofbirth": {
16      "data": null,
17      "display": true
18    },
19    "gender": {
20      "data": "",
21      "display": true
22    },
23    "Program": {
24      "data": "",
25      "display": true
26    },
27    "Description": {
28      "data": "",
29      "display": true
30    }
31  },
32  "result": 1,
33  "message": "User deleted."
34 }
```

- On failure

```
1 ▼ {
2 ▼   "data": {
3 ▼     "_id": {
4       "$oid": "633efce10cc286be9c9f6b9f"
5     },
6 ▼     "email": {
7       "data": "alfred@utoronto.ca",
8       "display": true
9     },
10    "password": "$2b$10$beEJ3TqgTvQTxhc4/yCFj./FFIPjA7b1GPF11bb7tgTVTwFeOHvXu",
11    "name": {
12      "data": "",
13      "display": true
14    },
15    "dateofbirth": {
16      "data": null,
17      "display": true
18    },
19    "gender": {
20      "data": "",
21      "display": true
22    },
23    "Program": {
24      "data": "",
25      "display": true
26    },
27    "Description": {
28      "data": "",
29      "display": true
30    }
31  },
32  "result": 0,
33  "message": "Error when deleting."
34 }
```

## POST /edit

**Description:** Edit an existing user's profile with the corresponding email passed by the body parameters from the student collection in the database, return a json file which contains 3 parts including the req (all body parameters), result of 0 (failure) /1 (success), and a message.

### Body Parameters:

- Required
  - Email: The email of the existing student.
  - Name, Gender, DateOfBirth, Program, Description: New data input by user.

### Expected Response:

- Body parameter example:

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, API Network, and Explore. The main area shows a collection named "My first collection" with two folders: "First folder inside collection" and "Second folder inside collection". Under "First folder inside collection", there are four GET requests. The "Body" tab is selected in the request configuration panel, showing a JSON payload:

```
{"_id": "633e067e800b5993d72f3af5", "email": {"data": "chau@utozonto.ca", "display": true}, "password": "chauchau", "name": {"data": "Chau Nguyen", "display": true}, "dateofbirth": {"data": null, "display": true}, "gender": {"data": "", "display": true}, "Program": {"data": "", "display": true}, "Description": {"data": "", "display": true}}
```

At the bottom, there are tabs for Body, Cookies, Headers, and Test Results. The status bar at the bottom right indicates a 200 OK response with a time of 246 ms and a size of 721 B.

- On success: Email address is registered in the database.

My Workspace

Overview POST http://localhost:8080/

POST http://localhost:8080/edit

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

2   "data": {
3     "_id": "633e067e800b5993d72f3af5",
4     "email": {
5       "data": "chau@utoronto.ca",
6       "display": true
7     },
8     "password": "$2b$10$BkbuaINq240Txqz7uNth3u7Rohu7yGntV9FytmbOK8A0810GBhK",
9     "name": {
10       "data": "Chau Nguyen",
11       "display": true
12     },
13     "dateofbirth": {
14       "data": null,
15       "display": true
16     },
17     "gender": {
18       "data": "",
19       "display": true
20     },
21     "Program": {
22       "data": "",
23       "display": true
24     },
25     "Description": {
26       "data": "",
27       "display": true
28     },
29   },
30   "result": 1,
31   "message": "Successfully updated user's profile."
32 }
```

Start working with APIs

67% Next: Save a request. Show me

Online Find and Replace Console Cookies Capture requests Bootcamp Runner Trash

- On failure: Email address is not registered in the database. Connection to database/server fails.

My Workspace

Overview POST http://localhost:8080/

POST http://localhost:8080/edit

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

2   "data": {
3     "_id": "633e067e800b5993d72f3af5",
4     "email": {
5       "data": "cha@utoronto.ca",
6       "display": true
7     },
8     "password": "chauchau",
9     "name": {
10       "data": "Chau Nguyen",
11       "display": true
12     },
13     "dateofbirth": {
14       "data": null,
15       "display": true
16     },
17     "gender": {
18       "data": "",
19       "display": true
20     },
21     "Program": {
22       "data": "",
23       "display": true
24     },
25     "Description": {
26       "data": "",
27       "display": true
28     },
29   },
30   "result": 0,
31   "message": "User does not exist."
32 }
```

Start working with APIs

67% Next: Save a request. Show me

Online Find and Replace Console Cookies Capture requests Bootcamp Runner Trash

## **POST/searchprogramskey**

**Description:** By using a string of keywords, filter out the programs in the database under program collection, Return a list of json files which contains the list of program type, list of program name and the length of the list that matches the searching key.

### **Body Parameters:**

- keywords: String

### **Expected Response**

If not found any similar programs:

```
1
2   "length": 0,
3   "result": []
4
```

If found similar programs:

```
1
2   "length": 2,
3   "result": [
4     {
5       "name": "Major (Co-operative) Program in Computer Science",
6       "type": "Major"
7     },
8     {
9       "name": "Major test",
10      "type": "Major"
11    }
12  ]
13
```

## **POST/searchprogramname**

**Description:** By using program name, find out one specific program in the database under program collection, Return json file that contains all the information about this specific program..

### **Body Parameters:**

- name: String

### **Expected Response**

If program name not found:

```
1
2   "result": 0,
3   "item": []
4
```

If found the specific program:

```
2 "result": 1,
3 "item": [
4     {
5         "_id": "634752535b8a89aac1586600",
6         "name": "Major test",
7         "type": "Major",
8         "area": "Computer Science",
9         "degree": "BSc",
10        "coop": "Students must satisfactorily complete three Co-op work terms, each of four-months duration, one of which can be duri
11        term, students must be enrolled in the Major (Co-op) Program in Computer Science and have completed at least 7.0 credits,
12        CSCA48H3, CSCA67H3, MATA22H3, MATA31H3, MATA37H3). In addition to their academic program requirements, Co-op students comp
13        are designed to prepare students for their job search and work term experience, and to maximize the benefits of their Co-o
14        to assist students in developing the skills and tools required to secure work terms that are appropriate to their program
15        workplace. These courses must be completed in sequence, and are taken in addition to a full course load. They are recorded
16        considered to be additive credit to the 20.0 required degree credits. No additional course fee is assessed as registration
17        Preparation Course Requirements: 1. COPB50H3/ (COPD01H3) - Foundations for Success in Arts & Science Co-op - Students ente
18        postsecondary) will complete this course in Fall or Winter of their first year at UTSC. Enrolment in each section is based
19        Science, Mathematics and Statistics enroll in the Fall semester while all other Arts & Science Co-op admission categories
20        year to year. - Current UTSC students entering Co-op in April/May will complete this course in the Summer semester. - Curr
21        complete this course in the Fall semester. 2. COPB51H3/ (COPD03H3) - Preparing to Compete for your Co-op Work Term - This
22        first scheduled work term. 3. COPB52H3/ (COPD11H3) - Managing your Work Term Search & Transition to Work - This course wil
23        scheduled work term. 4. COPC98H3/ (COPD12H3) - Integrating Your Work Term Experience Part I - This course will be complete
24        term. 5. COPC99H3/ (COPD13H3) - Integrating Your Work Term Experience Part I - This course will be completed four months i
25        programs that require the completion of 3 work terms and/or four months in advance of any additional work terms that have
26        Students must be available for work terms in each of the Fall, Winter and Summer semesters and must complete at least one
27        semester. This, in turn, requires that students take courses during at least one Summer semester.","
28        "enrolment": "Enrolment in the Major (Co-operative) Program in Computer Science is limited. Current Co-op Students: Students a
29        study must request a Co-op Subject POST on ACORN upon completion of 4.0 credits. Students must have completed the required
30        grades, described in the Enrolment Requirements for the Major in Computer Science. In addition, they must also have achiev
31        Prospective Co-op Students: Prospective students (i.e., those not yet admitted to a Co-op Degree POST) must meet the enrol
32        1. A minimum cumulative average of 2.75 across all attempted courses. Students must submit a program request on ACORN. Deadlines follow the Limited Enrolment
33        of the Registrar each year. Failure to submit the program request on ACORN will result in the student's application not being c
34        "graduation": "The course requirements of the Co-operative Major Program in Computer Science are identical to those of the Major Pi
35        program, students must maintain a CGPA of 2.5 or higher throughout the program. To complete the program, students must meet the
36        "description": "The Major (Co-op) Program in Computer Science is a Work Integrated Learning (WIL) program that combines academic st
37        and/or non-profit sectors. The program provides students with the opportunity to develop the academic and professional skills i
38        continue on to graduate training in an academic field related to Computer Science upon graduation.",
39        "note": null,
40        "status": "Available"
```

## POST/advanceprograms

**Description:** Using a json file which contains the information about the advance search, search in program collection to find if there's any matches. Return a Json object which contains every program object matched to the keywords.

**Body Parameters:**

```
1 ...
2     ...
3     "keywords": "COMPUTER SCIENCE",
4     ...
5     "area": "COMPUTER",
6     ...
7     "degree": "BA",
8     ...
9     "enrolment": "",
10    ...
11    "coop": "\\\W",
12    ...
13    "type": "SCIENCE"
```

## **Expected Response:**

If not found any matching programs:

```
Pretty Raw Preview visualize JSON ▾
```

1  
2     "length": 0,  
3     "finalresult": []  
4

If found matching programs:

## **POST /searchcourse**

**Description:** By using a string of keywords, search in the database under course collection, for every course that has a similar name to the keywords. Return a JSON object which contains every course object matched to the keywords.

#### **Body Parameters:**

- keywords: String (give keywords:"", will give result based on Breadth)
  - breadth: String (give Breadth:"", will disable Breadth and just search on keywords)  
If both keywords and Breadth give an empty string, will return all the courses in the DB.

### **Expected Response:**

If not found any similar course:

"check": 0

If found any similar course:

```
{
  "check": 1,
  "a": [
    {
      "name": "Africa in the World: An Introduction",
      "code": "AFSA01H3"
    },
    {
      "name": "Introduction to Software Engineering",
      "code": "CSCC01H3"
    },
    {
      "name": "Software Design",
      "code": "CSCB07H3"
    },
    {
      "name": "Software Tools and Systems Programming",
      "code": "CSCB09H3"
    },
    {
      "name": "Supervised Introductory Research in Psychology",
      "code": "PSYB90H3"
    },
    {
      "name": "Bioinorganic Chemistry",
      "code": "CHMD69H3"
    }
  ]
}
```

## POST /advanceSearch

**Description:** Based on the general search, users can input more filters to get the desired results.

### Body Parameters:

- keywords: String
- breadth: String
- description: String
- score: int
- pre: String
- level: String

Ex:

```
{
  "res": {
    "keywords": "",
    "breadth": "",
    "description": "",
    "score": {
      "average": 1
    },
    "level": "A",
    "pre": ""
  }
}
```

### Expected Response:

If not found any similar course:

```
{  
    "check": 0  
}
```

If found any similar course:

```
{  
    "check": 1,  
    "a": [  
        {  
            "name": "Africa in the World: An Introduction",  
            "code": "AFSA01H3"  
        }  
    ]  
}
```

## POST /sortSearch

**Description:** Based on the general search, users can input more filters to get the desired results.

### Body Parameters:

- keywords: String
- breadth: String
- description: String
- score: int
- pre: String
- level: String
- sort: int (1 for sort from small to big, 2 for reverse, 0 for no sort)

Ex:

```
{  
    "res": {  
        "keywords": "",  
        "breadth": "",  
        "description": "",  
        "score": {  
            "average": 0  
        },  
        "level": "",  
        "pre": "",  
        "sort": 2  
    }  
}
```

### Expected Response:

If not found any similar course:

```
{  
    "check": 0  
}
```

If found any similar course:

```
{  
    "check": 1,  
    "e": [  
        {  
            "name": "Africa in the World: An Introduction",  
            "code": "AFSA01H3",  
            "score": 6  
        },  
        {  
            "name": "Business of Journalism",  
            "code": "JOURB03H3",  
            "score": 6  
        },  
        {  
            "name": "Human Origins: New Discoveries",  
            "code": "ANTC17H3",  
            "score": 6  
        },  
        {  
            "name": "Foundation in Epidemiology",  
            "code": "ANTC67H3",  
            "score": 6  
        },  
        {  
            "name": "Contemporary African Studies",  
            "code": "AFSA17H3",  
            "score": 6  
        }  
    ]  
}
```

## POST /courseInfo

**Description:** By giving a course code, return all the information about that corresponding course.

### Body Parameters:

- code: String

### Expected Response:

If found:

```
    "check": 1,
    "course": {
        "_id": "63459124a6fe8574e54aaafdc",
        "code": "PSYB90H3",
        "name": "Supervised Introductory Research in Psychology",
        "description": "This course provides an introduction to, and experience in, ongoing theoretical and empirical research in any field of psychology. Supervision of the work is arranged by mutual agreement between student and instructor. Students will typically engage in an existing research project within a supervisor's laboratory. Regular consultation with the supervisor is necessary, which will enhance communication skills and enable students to develop proficiency in speaking about scientific knowledge with other experts in the domain. Students will also develop documentation and writing skills through a final report and research journal. This course requires students to complete a permission form obtained from the Department of Psychology. This form must outline agreed-upon work that will be performed, must be signed by the intended supervisor, and returned to the Department of Psychology.",
        "breadth": "Social and Behavioural Sciences",
        "exclusions": [
            "ROP299Y",
            "LINB98H3"
        ],
        "prerequisites": [
            "PSYA01H3 and PSYA02H3 with at least an 80% average across both courses",
            "a minimum of 4.0 credits [including PSYA01H3 and PSYA02H3] in any discipline, with an average cGPA of 3.0",
            "a maximum of 9.5 credits completed",
            "enrolment in a Psychology, Mental Health Studies, Neuroscience or Psycholinguistics program"
        ],
        "corequisites": [],
        "recommended": [
            "B-level courses in Psychology or Psycholinguistics"
        ],
        "notes": "1. Students receive a half credit spread across two terms, therefore, the research in this course must take place across two consecutive terms. 2. Priority will be given to students enrolled in a Specialist/Major program in Psychology or Mental health studies, followed by students enrolled in a Specialist/Major program in Neuroscience or Psycholinguistics. 3. Enrollment will depend each year on the research opportunities available with each individual faculty member and the interests of the students who apply.",
        "status": "Available"
    }
}
```

If not found:

```
1  {
2      "check": 0
3  }
```

## POST /course/rate

**Description:** Create a new document in the database for users' rating and comment on courses.

### Response Body:

```
{
    username: String,
    email: String,
```

```

course: String,
score: Integer,
comment: String,
anonymity: Boolean [default: true]
}

```

The screenshot shows the Postman interface with a collection named "Your collection". A POST request is being prepared to the endpoint `http://localhost:8080/course/rate`. The request body is set to `JSON` and contains the following data:

```

1  {
2     "username": "Chau Nguyen",
3     "email": "chau@utoronto.ca",
4     "course": "CSCB07H3",
5     "score": 5,
6     "comment": "very helpful",
7     "anonymity": true
8 }

```

### Expected Response:

```

{
  data: {
    rating: {
      Username: String,
      email: String,
      Anonymity: Boolean;
      created: Date,
      course: String,
      score: Integer,
      comment: String
    },
    comment: {
      Username: String,
      email: String,
      anonymity: Boolean
      created: Date,
      course: String,
      parent: String,
    }
  }
}

```

```

        content: String,
    }
},
result: Integer [0 or 1],
message: String
}

```

On success: User exists, username and email match. Course exists.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A 'Create a collection for your requests' section is also present. The main area shows an 'Overview' tab with a POST request to 'http://localhost:8080/course/rate'. The request details show 'POST' method, URL, and various tabs like Params, Authorization, Headers (8), Body (selected), Pre-request Script, Tests, and Settings. The Body tab displays a JSON response with status 200 OK, time 388 ms, and size 799 B. The response content is as follows:

```

1
2   "data": {
3     "rating": {
4       "username": "Chau Nguyen",
5       "email": "chau@utoronto.ca",
6       "anonymity": true,
7       "created": "2022-10-16T17:24:48.812Z",
8       "course": "CSCB07H3",
9       "score": 5,
10      "comment": "634c3e7bf8c5b5aa41c29f71",
11      "_id": "634c3e7bf8c5b5aa41c29f73",
12      "__v": 0
13    },
14    "comment": {
15      "username": "Chau Nguyen",
16      "email": "chau@utoronto.ca",
17      "anonymity": true,
18      "created": "2022-10-16T17:24:48.812Z",
19      "course": "CSCB07H3",
20      "parent": null,
21      "content": "very helpful",
22      "_id": "634c3e7bf8c5b5aa41c29f71",
23      "__v": 0
24    }
25  },
26  "result": 1,
27  "message": "Successfully posted new rating."
28

```

On failure:

- Missing username field

Home Workspaces API Network Explore

My Workspace

New Import

Overview POST http://localhost:8080/

No Environment

http://localhost:8080/course/rate

POST http://localhost:8080/course/rate

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON Cookies Beautify

```
1
2   "email": "chau@utoronto.ca",
3   "course": "CSCB07H3",
4   "score": 5,
5   "comment": "very helpful",
6   "anonymity": true
```

Body Cookies Headers (8) Test Results

Status: 400 Bad Request Time: 92 ms Size: 381 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2   "data": {
3     "rating": null,
4     "comment": null
5   },
6   "result": 0,
7   "message": "User does not exist."
```

Start working with APIs

67% Next: Save a request. Show me

Online Find and Replace Console Cookies Capture requests Bootcamp Runner Trash

- Missing email field

Home Workspaces API Network Explore

My Workspace

New Import

Overview POST http://localhost:8080/

No Environment

http://localhost:8080/course/rate

POST http://localhost:8080/course/rate

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON Cookies Beautify

```
1
2   "username": "Chau Nguyen",
3   "course": "CSCB07H3",
4   "score": 5,
5   "comment": "very helpful",
6   "anonymity": true
```

Body Cookies Headers (8) Test Results

Status: 400 Bad Request Time: 90 ms Size: 381 B Save Response

Pretty Raw Preview Visualize JSON

```
1
2   "data": {
3     "rating": null,
4     "comment": null
5   },
6   "result": 0,
7   "message": "User does not exist."
```

Start working with APIs

67% Next: Save a request. Show me

Online Find and Replace Console Cookies Capture requests Bootcamp Runner Trash

## - Course code does not exist

The screenshot shows the Postman interface with a failed API request. The request is a POST to `http://localhost:8080/course/rate`. The body contains the following JSON:

```
1 {
2     "username": "Chau Nguyen",
3     "email": "chau@utoronto.ca",
4     "course": "CSCB07",
5     "score": 5,
6     "comment": "very helpful",
7     "anonymity": true
8 }
```

The response status is 400 Bad Request, with the message: "Course does not exist."

## - Missing course code field

The screenshot shows the Postman interface with a failed API request. The request is a POST to `http://localhost:8080/course/rate`. The body contains the following JSON:

```
1 {
2     "username": "Chau Nguyen",
3     "email": "chau@utoronto.ca",
4     "score": 5,
5     "comment": "very helpful",
6     "anonymity": true
7 }
```

The response status is 400 Bad Request, with the message: "Course does not exist."

## - Missing comment field

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' with sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A 'Create a collection for your requests' section is also present. The main workspace shows a POST request to 'http://localhost:8080/course/rate'. The 'Body' tab is selected, displaying the following JSON payload:

```
1 "username": "Chau Nguyen",
2 "email": "chau@utoronto.ca",
3 "course": "CSCB07H3",
4 "score": 5,
5 "anonymity": true
```

The 'Test Results' tab shows a 400 Bad Request response with the following JSON body:

```
1 "data": {
2     "rating": null,
3     "comment": null
4 },
5 "result": 0,
6 "message": "Comment validation failed: content: Content required."
```

At the bottom, a progress bar indicates 'Start working with APIs' at 67%, with a link to 'Show me'.

## - Missing score field

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, API Network, Explore, Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area shows a collection named "Your collection" with one item. A modal window is open for a POST request to "http://localhost:8080/course/rate". The request body is set to "JSON" and contains the following JSON:

```

1
2   ...
3     "username": "Chau Nguyen",
4     "email": "chau@utoronto.ca",
5     "course": "CSCB07H3",
6     "comment": "very helpful",
7     "anonymity": true
8

```

The response tab shows a 400 Bad Request status with the following JSON message:

```

1
2   ...
3     "data": {
4       "rating": null,
5       "comment": null
6     },
7     "result": 0,
8     "message": "Rating validation failed: score: Score required."
9

```

## POST /seeCourseRatings

**Description:** By giving a course, return all the ratings and each rating's corresponding comment (if any).

### Body Parameters:

- Required
  - Course: String

### Expected Response:

- Return a json file containing all the returning data
- Body parameter example:

The screenshot shows the Postman interface with the 'Body' tab selected. The 'raw' option is chosen, and the JSON content is:

```
1 {"course":"CSCC01H3"}
```

- Returned json file for the body parameter example:

The screenshot shows the JSON response in Postman. The response structure is as follows:

```
1 {
2   "result": 1,
3   "ratings": [
4     {
5       "_id": "63480a6dbd7bfae32073d358",
6       "email": "chau@utoronto.ca",
7       "anonymity": true,
8       "created": "2022-10-13T12:53:29.644Z",
9       "course": "CSCC01H3",
10      "score": 10,
11      "comment": "63480a6cbd7bfae32073d356",
12      "__v": 0
13    },
14    {
15      "_id": "634840d017ab8d442c4258f9",
16      "email": "chau@utoronto.ca",
17      "anonymity": true,
18      "created": "2022-10-13T16:44:03.474Z",
19      "course": "CSCC01H3",
20      "score": 5,
21      "comment": "634840d017ab8d442c4258f7",
22      "__v": 0
23    },
24    {
25      "_id": "63487a8a1eb73ec8153c41b0",
26      "email": "hpttesting.com",
27      "anonymity": true,
28      "created": "2022-10-13T20:47:47.789Z",
29      "course": "CSCC01H3",
30      "score": 5,
31      "comment": "63487a891eb73ec8153c41ae",
32      "__v": 0
33    }
34  ],
35  "comments": [
36    {
37      "id": "63480a6cbd7bfae32073d356",
38      "comment": "This is a test comment"
39    }
40  ]
41 }
```

```

35 ▾      "comments": [
36 ▾          {
37              "_id": "63480a6cbd7bfae32073d356",
38              "email": "chau@utoronto.ca",
39              "anonymity": true,
40              "created": "2022-10-13T12:53:29.644Z",
41              "course": "CSCC01H3",
42              "parent": null,
43              "content": "test",
44              "__v": 0
45          },
46          {
47              "_id": "634840d017ab8d442c4258f7",
48              "email": "chau@utoronto.ca",
49              "anonymity": true,
50              "created": "2022-10-13T16:44:03.474Z",
51              "course": "CSCC01H3",
52              "parent": null,
53              "content": "i don't recommend this course to anyone wishing to have enough sleep",
54              "__v": 0
55          },
56          {
57              "_id": "63487a891eb73ec8153c41ae",
58              "email": "hpttesting.com",
59              "anonymity": true,
60              "created": "2022-10-13T20:47:47.789Z",
61              "course": "CSCC01H3",
62              "parent": null,
63              "content": "hello",
64              "__v": 0
65          }
66      ],
67      "message": "returning course ratings."
68  }

```

- result: 1 on success, 0 on failure

Note: failure only occurred for database issues for fetching data. In other words, even if a course has no ratings nor any corresponding comments, it would return a json file with an empty list of ratings, and an empty list of comments. However, the result would still be 1 indicating success.

i.e.

```

1 ▾  {
2      "result": 1,
3      "ratings": [],
4      "comments": [],
5      "message": "returning course ratings."
6  }

```

**POST /comment**

**Description:** Create a child comment in the database.

**Request body:**

```
{  
    "email": String (email of user) (required),  
    "course": String (course code for the course being commented on) (required),  
    "id": String (id of parent comment) (default is null, meaning this is a root comment  
with no parent),  
    "comment": String (content of the comment) (required),  
    "anonymity": Boolean (whether user wants their name displayed to others)  
(default is true)  
}
```

The screenshot shows the Postman application interface. On the left, the sidebar has sections for Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A collection named 'My first collection' is expanded, showing two folders: 'First folder inside collection' and 'Second folder inside collection', each containing a single GET request. Below the collection is a section titled 'Create a collection for your requests' with a description of what collections are used for. In the main workspace, there is a 'POST' request to 'http://localhost:8080/comment'. The 'Body' tab is selected, showing the following JSON code:

```
1  ... "email": "chau@utoronto.ca",  
2  ... "course": "CSCC01H3",  
3  ... "id": "6350d064f5a5c59f7048f17",  
4  ... "comment": "I agree. This course was such a nice addition to my resume.",  
5  ... "anonymity": false
```

At the bottom of the interface, there is a progress bar labeled 'Start working with APIs' at 67%, and a status bar showing 'Status: 200 OK Time: 276 ms Size: 959 B Save Response'.

**Expected Response:**

```
{  
    "data": {  
        "parent-comment": Object (all fields of parent comment),  
        "child-comment": Object (all fields of new comment)  
    },  
    "result": 0/1 (0 for failure, 1 for success),  
    "message": String (error/success message)  
}  
Success
```

Home Workspaces API Network Explore

My Workspace New Import

Overview POST http://localhost:8080/

http://localhost:8080/comment

POST http://localhost:8080/comment

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

1   "data": {
2     "parent-comment": {
3       "_id": "63505d064f5a5c59f7048f17",
4       "email": "Cale@utoronto.ca",
5       "anonymity": false,
6       "created": "2022-10-19T20:10:43.010Z",
7       "course": "CSCC01H3",
8       "parent": null,
9       "content": "Love the course!",
10      "---v": 1,
11      "numDislikes": 20,
12      "numLikes": 17
13    },
14    "child-comment": {
15      "email": "chau@utoronto.ca",
16      "anonymity": false,
17      "created": "2022-11-02T21:42:45.929Z",
18      "course": "CSCC01H3",
19      "parent": "63505d064f5a5c59f7048f17",
20      "content": "I agree. This course was such a nice addition to my resume.",
21      "likedEmails": [],
22      "dislikedEmails": [],
23      "numLikes": 0,
24      "numDislikes": 0,
25      "_id": "6362e4c73ad077a7976e2b51",
26      "---v": 0
27    }
28  },
29  "result": 1,
30  "message": "Successfully posted new comment."
31

```

Start working with APIs

67% Next: Save a request. Show me

Online Find and Replace Console

Upgrade

## Failure: missing email

Home Workspaces API Network Explore

My Workspace New Import

Overview POST http://localhost:8080/

http://localhost:8080/comment

POST http://localhost:8080/comment

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

1   "course": "CSCC01H3",
2   "id": "63505d064f5a5c59f7048f17",
3   "comment": "I agree. This course was such a nice addition to my resume.",
4   "anonymity": false
5
6

```

Start working with APIs

67% Next: Save a request. Show me

Online Find and Replace Console

Upgrade

## Failure: missing parent comment id

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' with 'Collections', 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. A central panel displays a collection named 'My first collection' containing 'First folder inside collection' and 'Second folder inside collection', each with several GET requests. Below this is a section titled 'Create a collection for your requests' with a 'Create collection' button. The main workspace shows a POST request to 'http://localhost:8080/comment'. The 'Body' tab is selected, showing the following JSON payload:

```
1
2   ...
3     "email": "chau@utoronto.ca",
4     ...
5       "course": "CSCC01H3",
6       ...
7         "comment": "I agree. This course was such a nice addition to my resume.",
8         ...
9           "anonymity": false
```

The response status is 400 Bad Request, with the message: "data": { "parent-comment": null, "child-comment": null }, "result": 0, "message": "Parent does not exist.".

## Failure: wrong comment id

This screenshot shows the same Postman interface as the previous one, but with a different JSON payload in the body. The 'Body' tab shows:

```
1
2   ...
3     "email": "chau@utoronto.ca",
4     ...
5       "course": "CSCC01H3",
6       ...
7         "id": "63505d064f5a5c59f7048f",
8         ...
9           "comment": "I agree. This course was such a nice addition to my resume.",
10          ...
11            "anonymity": false
```

The response status is 400 Bad Request, with the message: "data": { "parent-comment": null, "child-comment": null }, "result": 0, "message": "Cast to ObjectId failed for value \"63505d064f5a5c59f7048f\" (type string) at path \"\_id\" for model \"Comment\"".

## Failure: missing course

The screenshot shows the Postman interface with a failed API request. The request is a POST to `http://localhost:8080/comment`. The body contains:

```
1 "email": "chau@utoronto.ca",
2 "id": "63505d064f5a5c59f7048f17",
3 "comment": "I agree. This course was such a nice addition to my resume.",
4 "anonymity": false
```

The response status is 400 Bad Request, with the message: "Comment validation failed: course: Course required."

## Failure: missing content of comment

The screenshot shows the Postman interface with a failed API request. The request is a POST to `http://localhost:8080/comment`. The body contains:

```
1 "email": "chau@utoronto.ca",
2 "course": "CSCC01H3",
3 "id": "63505d064f5a5c59f7048f17",
4 "anonymity": false
```

The response status is 400 Bad Request, with the message: "Comment validation failed: content: Content required."

## POST /seeRatingComments

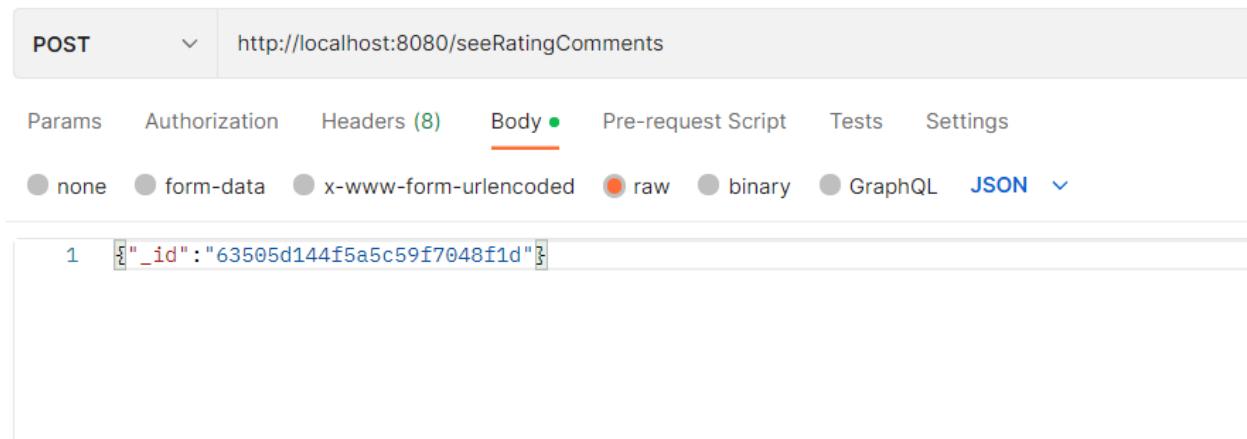
**Description:** By giving an id of the parent comment, return all the comments that are child comments of the parent comment (i.e. all comments that have the id of the parent comment in their parent field).

### Body Parameters:

- Required
  - \_id: String

### Expected Response:

- Return a json file containing all the returning data
- Body parameter example:



```
1 {  
2     "_id": "63505d144f5a5c59f7048f1d"  
3 }  
  
1 {  
2     "result": 1,  
3     "child comments": [  
4         {  
5             "_id": "6362eed9e0b2e6e80fe517c9",  
6             "email": "Caleb@utoronto.ca",  
7             "anonymity": true,  
8             "created": "2022-11-02T22:04:57.029Z",  
9             "course": "CSCC01H3",  
10            "parent": "63505d144f5a5c59f7048f1d",  
11            "content": "I agree!",  
12            "likedEmails": [],  
13            "dislikedEmails": [],  
14            "numLikes": 0,  
15            "numDislikes": 0,  
16            "__v": 0  
17        },  
18    ],  
19    "names": [  
20        "Keyuan Zhang"  
21    ]  
}
```

## POST /like

**Description:** By giving an id of the comment and the email of the user, the user will

- Like the comment if user has not liked or disliked the comment before
  - In database it will +1 to numLikes
  - In database it will add the user email to the likedEmails array
- Cancel like comment if user has liked the comment before
  - In database it will -1 to numLikes
  - In database it will remove the user email in the likedEmails array
- Cancel dislike and like comment if the user has disliked the comment before
  - In database it will +1 to numLikes and -1 to numDislikes
  - In database it will add the user email to the likedEmails array and remove the user email in the dislikedEmails array

### Body Parameters:

- Required
  - \_id: String
  - email: String

### Expected Response:

- Return a json file containing all the returning data
- Result 1: if user liked
- Result 2: if user cancel like
- Result 3: if user cancel dislike and liked

The screenshot shows a Postman request configuration. The method is set to POST, and the URL is http://localhost:8080/like. The 'Body' tab is selected, showing a JSON payload with the key '\_id' and value '63508b0ae3c267c8859a6b2d', and the key 'email' and value 'alfred.ha@mail.utoronto.ca'. The 'Pretty' tab is selected in the response view, displaying the following JSON output:

```
1 {  
2   "result": 1,  
3   "likes": 2,  
4   "dislikes": 13  
5 }
```

```
1  {"_id": "63508b0ae3c267c8859a6b2d", "email": "alfred.ha@mail.utoronto.ca"}  
  
Body Cookies Headers (9) Test Results 🌐 200 OK  
  
Pretty Raw Preview Visualize JSON 🔗  
  
1  {  
2      "result": 2,  
3      "likes": 1,  
4      "dislikes": 13  
5  }
```

```
POST http://localhost:8080/like  
  
Params Authorization Headers (8) Body Pre-request Script Tests Settings  
none form-data x-www-form-urlencoded raw binary GraphQL JSON  
  
1  {"_id": "63508b0ae3c267c8859a6b2d", "email": "alfred.ha@mail.utoronto.ca"}  
  
Body Cookies Headers (9) Test Results 🌐 200 OK 9  
  
Pretty Raw Preview Visualize JSON 🔗  
  
1  {  
2      "result": 3,  
3      "likes": 2,  
4      "dislikes": 13  
5  }
```

## POST /dislike

**Description:** By giving an id of the comment and the email of the user, the user will

- Dislike the comment if user has not disliked or liked the comment before
  - In database it will +1 to numDislikes
  - In database it will add the user email to the dislikedEmails array
- Cancel dislike comment if user has disliked the comment before
  - In database it will -1 to numDislikes
  - In database it will remove the user email in the dislikedEmails array
- Cancel like and dislike comment if the user has liked the comment before
  - In database it will +1 to numDislikes and -1 to numLikes
  - In database it will add the user email to the dislikedEmails array and remove the user email in the likedEmails array

**Body Parameters:**

- Required
  - \_id: String
  - email: String

**Expected Response:**

- Return a json file containing all the returning data
- Result 1: if user disliked
- Result 2: if user cancel dislike
- Result 3: if user cancel like and disliked

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' selected as the method and 'http://localhost:8080/dislike' as the URL. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is currently active, indicated by a green dot. Under the 'Body' tab, there are several options: 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected and highlighted in orange), 'binary', and 'GraphQL'. To the right of these options is a 'JSON' dropdown menu. Below the tabs, there is a code editor area containing a JSON object:

```
1 { "id": "63508b0ae3c267c8859a6b2d", "email": "alfred.ha@mail.utoronto.ca" }
```

At the bottom of the interface, there are more tabs: 'Body', 'Cookies', 'Headers (9)', and 'Test Results'. The 'Test Results' tab is currently active, indicated by a red underline. To the right of the tabs, there is a status indicator showing a globe icon and the number '200'. Below the tabs, there are four buttons: 'Pretty', 'Raw', 'Preview', and 'Visualize'. The 'Pretty' button is selected and highlighted in grey. To the right of these buttons is a 'JSON' dropdown menu.

Under the 'Pretty' button, there is a code editor area showing the JSON response:

```
1 {  
2   "result": 3,  
3   "likes": 1,  
4   "dislikes": 14  
5 }
```

POST  http://localhost:8080/dislike

Params Authorization Headers (8) **Body** • Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 { "_id": "63508b0ae3c267c8859a6b2d", "email": "alfred.ha@mail.utoronto.ca" }
```

Body Cookies Headers (9) Test Results

200 OK

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "result": 2,
3   "likes": 1,
4   "dislikes": 13
5 }
```

POST  http://localhost:8080/dislike

Params Authorization Headers (8) **Body** • Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 { "_id": "63508b0ae3c267c8859a6b2d", "email": "alfred.ha@mail.utoronto.ca" }
```

Body Cookies Headers (9) Test Results

200 OK

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "result": 1,
3   "likes": 1,
4   "dislikes": 14
5 }
```

### **POST/getrecommend**

**Description:** By receiving the list of courses the student completed and the preference of the student, get the recommended courses for the student to complete the program

**Request Body:**

```
{  
    ... "stream": "Entrepreneurship",  
    ... "preference": "python|Java|C++",  
    ... "completed": ["csca08h3", "csca48h3"]  
}
```

**Response Body:**

```
8  
9 ↴ [  
.0   ... "completed": ["csca08h3", "csca48h3"],  
.1   ... "required": {"A": [], "B": [], "C": [], "D": []},  
.2   ... "electives": {"A": [], "B": [], "C": [], "D": []}  
.3 ]
```

### **POST /course/list**

**Description:** Get a list of course codes in the database

**Request Body:** not required

**Response Body:**

```
{  
    "courses": [  
        "AFSA01H3",  
        "CSCC01H3",  
        "CSCB07H3",  
        "CSCB09H3",  
        "PSYB90H3",  
        "JOUB03H3",  
        "ACMA01H3",  
        "ANTC17H3",  
        "ANTC67H3",  
        "ANTC99H3",  
        "ASTC25H3",  
        "BIOC29H3",  
        "BIOD22H3",  
        "CHMA11H3",  
        "CHMD91H3",  
        "CLAB05H3",  
        "COPC03H3",
```

"EESB26H3",  
"EESD19H3",  
"ENGB33H3",  
"ENGС13H3",  
"ENGС44H3",  
"ENGD03H3",  
"ENGD71H3",  
"FREA98H3",  
"FREC38H3",  
"FSTD10H3",  
"GASC73H3",  
"GRC01H3",  
"GGRD08H3",  
"HISB30H3",  
"HISC10H3",  
"HISC70H3",  
"HISC48H3",  
"HLTB40H3",  
"HLTC55H3",  
"CTLA21H3",  
"CHMD69H3",  
"CSCA08H3",  
"CSCA48H3",  
"CSCA67H3",  
"MATA67H3",  
"MATA22H3",  
"MATA31H3",  
"MATA37H3",  
"STAB52H3",  
"MATB24H3",  
"CSCB63H3",  
"CSCB58H3",  
"CSCB36H3",  
"CS СC43H3",  
"CS СC69H3",  
"CS СC73H3",  
"CSCD03H3",  
"CS СC24H3",  
"CS СC37H3",  
"CSCD37H3",  
"MATB41H3",  
"MATA23H3",  
"MATA36H3",

```
"MATA30H3",
"MATA32H3",
"MATA33H3",
"CSCA65H3",
"PHYB57H3",
"CSCC09H3",
"CSCC10H3",
"CSCC11H3",
"CSCC46H3",
"CSCC85H3",
"CSCD01H3",
"CSCD25H3",
"MATD16H3",
"MATC44H3",
"MATC32H3",
"MATB43H3",
"MATC09H3",
"CSCD84H3",
"CSCD70H3",
"CSCD58H3",
"CSCD43H3",
"CSCD27H3",
"CSCD25H3",
"MGTA01H3",
"MGTA02H3",
"MGHB02H3",
"MGTA05H3",
"MGTA35H3",
"MGTA36H3",
"CSCD54H3",
"CSCD90H3"
]
}
```

## **POST /checkPrerequisites**

**Description:** By giving a course code and the user's academic history, check if the user satisfies the requirements to take the course with the course code based on the user's academic history

### **Body Parameters:**

- Required
  - code: String
  - student\_academic\_history: String

### **Expected Response:**

- Return a json file containing all the returning data including the following:
  - result
    - An integer with value:
      - 0: Error occurred
      - 1: User is able to take the course
      - 2: User is missing prerequisite(s) to take the course
      - 3: User has already took the course or exclusion(s) of the course
    - Prerequisites
      - An array containing all the missing prerequisite(s) if the user is missing prerequisite(s) to take the course
      - An empty array otherwise
    - message
      - A string that summarized the result (i.e. the 4 possibly results - 0: Error occurred, 1: User is able to take the course, 2: User is missing prerequisite(s) to take the course, 3: User has already took the course or exclusion(s) of the course)

# Frontend Documentation

### **How to update React dependencies and start frontend:**

```
$ npm install  
$ npm start
```

### **File Structure**

- The file /shortcut/frontend/src/routes.js clarifies all routes between pages.
- The folder /shortcut/frontend/src/Components contains frontend elements which are defined separately with respective .js and .css files.
- The folder /shortcut/frontend/src/Images contains image files that are/will be used in pages.

- The folder /shortcut/frontend/src/Pages contains folders for distinct pages. In the folders are .js and .css files for respective pages.

## Page Summaries

- **Login:** This is the opening page of the website. An existing user can sign in with the correct email-password combination. The user will be sent to the Home page upon a successful login operation. A new user can navigate to the Signup page through a link.
- **Signup:** This page allows new users of the website to create a new account. The operation will be interrupted if the users provide invalid email/password inputs or fail to confirm the password. Users can go back to the Login page with a button.
- **Home:** This is the central page of the website. It allows users to navigate to the main sections of the website. By the end of sprint 4, users will be able to navigate to PersonalProfile, GeneralCourse, GeneralProgram, GradReq and History pages.
- **Profile:** This page allows users to view and edit their personal information. Users can choose to sign out or delete their accounts, after which they will be sent back to the Login page if the operation is successful. Users can also go back to the Home page through a button.
- **CourseDescription:** This page displays the course description. The page contains the average rate given by previous students. It also allows users to navigate to the respective CommentForm and CommentView pages: to make new comments or view other's comments.
- **GeneralCourse:** This page allows searching for courses with keywords and provides proper associations. There is an advanced search button on side that direct users to search in more fields. The advanced search allow users to search more detail on their desired field. Once the content is clicked, the user will be directed to the respective CourseDescription page.
- **ProgramDescription:** This page displays the program information. Including: descriptions, requirements, enrolments, etc.
- **GeneralProgram:** This page allows searching for programs with keywords and provides proper associations. There is an advanced search button on side that direct users to search in more fields. The advanced search allow users to search with their preference and find the most match course. Once the content is clicked, the user will be directed to the respective ProgramDescription page.
- **CommentForm:** This page comes after a user intends to create a new rating on a course description page. The user can input a score and a piece of comment and decide

whether to publish it anonymously. In either case of canceling or submitting the form, the user will return to the course description page.

- **CommentView:** This page allows a user to view all ratings and comments about the course. The user can like/dislike/comment on other students' comments. The user can return to the course page.
- **ChildCommentView:** This page allows the user to see the comments of a specific comment. The user can like/dislike on the comment of comments. This page is similar to CommentView.
- **ChildCommentForm:** This page is a form that allows the user to comment on others' comments. The form is similar to CommentForm.
- **Result:** This page shows user the recommended result from their submitted form. It illustrates all courses that user already completed, and shows the required/recommend courses based on their preference.
- **GradReq:** This page displays the general graduation requirements, mainly the degree requirements, in a natural form.
- **History:** This page requires users to enter a stream, a list of preferences and completed courses to reflect their academic history. The information will help generate the recommendation that will be used in the Result page, to which the users will be directed upon the submission of the form.
- Upcoming pages...

### Important Utils

Following packages are extra installed while implementing frontend functionality:

```
$ npm install react-router-dom@6
$ npm install react-datepicker
$ npm install react-icons
$ npm install @mui/material @emotion/react @emotion/styled
$ npm install @mui/material @mui/lab
```

## Database Schema

```
Student {
    email: [data: String, display: Boolean],
    password: String,
    dateofbirth: [data: String, display: Boolean],
    gender: [data: String, display: Boolean],
    Program: [data: String, display: Boolean],
    Description: [data: String, display: Boolean]
}
Course {
```

```
        code: String,  
        name: String,  
        description: String,  
        breadth: String  
        exclusions: [0(or)/1(and), String/Array of the same structure] or [],  
        prerequisites: [0(or)/1(and), String/Array of the same structure] or [],  
        corequisites: [0(or)/1(and), String/Array of the same structure] or [],  
        recommended: [0(or)/1(and), String/Array of the same structure] or [],  
        note: String,  
        status: String,  
        score: {average: Number, num: Number}  
    }  
Program {  
    name: String,  
    type: String,  
    area: String,  
    degree: String,  
    coop: String,  
    enrolment: String,  
    graduation: String,  
    description: String,  
    note: String,  
    status: String (offered,not offered)  
}  
Rating {  
    email: String,  
    created: Date,  
    course: String,  
    score: Integer,  
    comment: String,  
    anonymity: Boolean,  
    difficulty: Integer  
}  
Comment {  
    email: String,  
    created: Date,  
    parent: String,  
    content: String,  
    anonymity: Boolean,  
    likedEmails: [String],  
    dislikedEmails: [String],  
    numLikes: Number,  
    numDislikes: Number  
}
```

