Link to Website: https://radiant-savannah-34193.herokuapp.com/

Link to GitHub: https://github.com/g5qinyix/CSC309project

Design of the application:

We chose to adopt a MVC (Model View Controller) architecture because it is simple in understanding how data is stored, obtained, changed, and transported within the context of our web application. It provides good organization, as it creates discrete boundaries between our model, view, and controller.

Model (in app/models):

The model pertains to our database and schema. We chose to utilize mongodb as our database management system because it is straightforward to add and remove entries; we do not need to learn SQL. In app/models, we have schemas for Users, Comments, and Messages. Attributes in Users pertain to their personal information (email, encrypted password, nickname, etc), whether they are a coach or a student, their preferred game, and their booking schedule. "Comments" refers to students posting comments on a coach's profile. It contains the date of the comments, studentid, and content of the comment. Messages have a receiver and sender.

View (in views):

The view contains .ejs files (similar to .html files), .css files to style the .ejs files, and JavaScript files that utilize jQuery to enhance the visual appearance of the website. It also includes features that check to ensure that the user enters valid information (i.e. email input should have @ followed by a .com/.ca).

Controller:

The controller takes data from the database/model and hands it to the view to be displayed for the user. Our main controller file is routes.js. It renders all the front-end web pages when needed; it routes the user to the proper web page, loading up any idiosyncratic information (for the user) obtained from the database. We also utilized passport to authenticate users (located in app/config/passport.js) via Facebook.

Each part of the web application and its interactions with other parts:

- There are two types of users for our web application: students and coaches. Coaches can choose to offer online coaching (coaching over Skype, or some other real-time chat software), offline coaching (choosing to meet up with the student at real life locations, like an internet café, to provide a more personal coaching experience), or both offline and online.
- Student Sign-up/Login: The user can sign up as a student or a coach. As a student, the user only needs to fill out some basic information (email, password, nickname, game, profile picture). After signing up, the user can sign in, find coaches, view coaches, view a specific coach's profile, send a message to a coach, schedule a coaching session, send money to the coach, follow a coach, and leave a rating for a coach. Alternatively, the user can sign up using Facebook. We used the Passport framework for Facebook sign-ins. The inputs from the students are checked using Sanitizer.

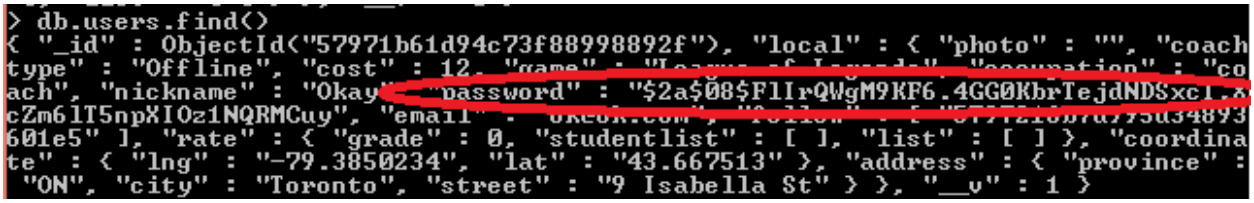
- Coach Sign-up/Login: A user can also sign up as a coach. They need to provide all the information that a student needs to as well as additional information: type of coaching offered (offline or online), address and location (only if they choose to offer offline coaching), the game the coach wishes to teach, and the price per hour the coach wishes to charge. After the coach signs up, he can view any messages sent to him by potential student users, read comments left on his profile, view his bookings, and view and find other coaches if HE wishes to partake as a student for a game. The inputs from coaches are checked using Sanitizer.
- Editing User Profile: Both students and coaches have the option of editing their profiles. They can change their password, nickname, and location. Coaches can also choose to change their coach type (offline/online) and price they wish to charge. The inputs are checked with Sanitizer and then replace the old values in the database.
- Coach Recommendation: Students who are looking for coaches can click on "Locate a Coach" and select a game. Afterwards, the student is presented with recommended coaches. The recommended coaches are displayed based on their ratings (highest ratings are showed).
- Offline Coach Searching: Students are able to search for coaches offering offline coaching. We've chosen to use Google Maps to display markers on a real map of Toronto. The markers on the map of Toronto displays an information window that contains the information of coaches. Each information window also links to the coach's profile. From there, the student can send a message to the coach and request a coaching session.
- Online Coach Searching: Students can also look for coaches that offer online coaching. The search system is similar to that of offline coaching. However, instead of seeing a map, the user sees display pictures of the coaches. The user can then click on the coach's picture to go to their profile.
- Messaging System: A student can message a coach about purchasing a potential coaching session by clicking on the quote bubble icon on the coach's profile. A coach can view all his/her messages by clicking on the mail icon at the top right hand corner of the web application. He can also reply to messages in the interface shown after clicking mail icon in the top right corner.
- Rating System: A coach needs to have a MINIMUM of three ratings from DIFFERENT students before a grade is shown. If the coach has fewer than three ratings, his rating will be: N/A. This is to ensure a more accurate representation of the coach's quality of teaching.
- Booking System: A student can request to book a coaching session by clicking on "order" from the coach's profile. Afterwards, the student will be presented with a timetable. The user can select a time(s) they wish to have coaching for. Afterwards, the user must click the "confirm" button at the bottom to send the request in. After the user clicks "confirm", money will be given to the coach.
- Payment System: After the user clicks the "confirm" button on the coach's timetable, money will be given to the coach, based on the number of hours booked. A message will also be sent to the coach to notify him of the coaching session.
- Following/Friend System: A student can follow a coach. The coach will then be added to the student's list of friends. The student can view the coaches he follow by clicking on the picture associated with the text "Your Favorite Coaches". The student can then click on any coach in the "following" view to go to their profile.
- Admin Functionality: If the admin logs-in with the user-email: admin@bsmaster.com and the password: admin. To access the admin page, the user can go to https://radiant-savannah-

34193.herokuapp.com/admin. The admin panel allows the admin to change the admin password, view all user accounts, add a user, update any user's information, delete any user, display all comments, delete any comment, display and delete all messages, delete all comments, and delete all users.

Potential Security Vulnerabilities:

We utilized Sanitizer to potential combat cross-site scripting: we "sanitized" all possible inputs from the user. If a user attempts to enter a dangerous tag (i.e. "<script> evilFunction(); </script>", the tag and contents are ignored. If the user enters an invalid tag, the tags are removed.

We also opted to use bcrypt that generates a hash to encrypt the user's password before storing said user's information in mongodb. This makes it so that even if a malicious hacker obtains the passwords somehow, the passwords are encrypted, and they of are no use to the hacker.



Finally, we used Google's reCAPTCHA to ensure that bots are unsuccessful when they attempt to brute-force their way into accessing a user's account (the reCAPTCHA may only work if run using 127.0.0.1:3000. There were some issues with the reCAPTCHA api key with the heroku hosted website).

Measures taken to improve the web application's performance:

We used node-load-tester to test our application's performance. We focused on the average time required for the server to send data to the client when a user logs in. In order to improve performance, we first tried to improve the performance on the front-end: we deleted unnecessary CSS code to reduce the size of the CSS files, and we utilized GZIP to compress data sent and http responses.

We also attempted to improve the performance on the back-end side. We used mongodb as our database management system, and we redesigned our database schema: at first, the collections have a list of messages. We created a new schema for messages, which contains the ids of the sender and receiver. The difference in our test results is not conspicuously obvious; the average response time in our initial test is always 9ms. The improvements resulted in an average response time of 8ms most of the time.

```
Process:
{
  "baseUrl": "http://localhost:3000",
  "duration": 5000,
  "connections": 1,
  "sequence": [
    { "method": "GET",  "path": "/" },
```

```
    { "method": "GET",  "path": "/login", "expect": { "code": 200 } },
    { "method": "POST", "path": "/login", "form": { "email": "1@mail.com"
,"password" : "19941026"} , "expect" : 302 },
    { "method": "GET",  "path": "/signup", "expect": { "code": 200 } },
    { "method": "GET",  "path": "/games/lol", "expect": { "code": 200 }},
    { "method": "GET",  "path": "/profile", "expect": { "code": 200 }},
    { "method": "GET",  "path": "/editstudent" ,"expect": { "code": 200
}},
    { "method": "GET",  "path": "/editcoach" ,"expect": { "code": 200 }},
    { "method": "GET",  "path": "/friend" ,"expect": { "code": 200 }},
    { "method": "GET",  "path": "/messaging", "expect": { "code": 200 }},
    { "method": "GET",  "path": "/logout" ,"expect": { "code": 302 }}
  ]
}

Initial test: Result:
{
  "paths": {
    "GET /": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 4
    },
    "GET /login": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 4
    },
    "POST /login": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 59
    },
    "GET /signup": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 4
    },
    "GET /games/lol": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 6
    },
```

```
    "GET /profile": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 4
    },
    "GET /editstudent": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 4
    },
    "GET /friend": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 5
    },
    "GET /messaging": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 7
    },
    "GET /logout": {
      "pass": 51,
      "fail": 0,
      "total": 51,
      "avgResponseTime": 3
    }
  },
  "errors": {},
  "pass": 510,
  "fail": 0,
  "total": 510,
  "totalTime": 5066,
  "avgResponseTime": 10
}
```


Improved results:


```
{
  "paths": {
    "GET /": {
      "pass": 56,
      "fail": 0,
      "total": 56,
```

```
  "avgResponseTime": 3
},
"GET /login": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 3
},
"POST /login": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 58
},
"GET /signup": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 3
},
"GET /games/lol": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 5
},
"GET /profile": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 4
},
"GET /editstudent": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 3
},
"GET /friend": {
  "pass": 56,
  "fail": 0,
  "total": 56,
  "avgResponseTime": 4
},
"GET /messaging": {
  "pass": 56,
  "fail": 0,
  "total": 56,
```

```
      "avgResponseTime": 6
    },
    "GET /logout": {
      "pass": 56,
      "fail": 0,
      "total": 56,
      "avgResponseTime": 3
    }
  },
  "errors": {},
  "pass": 560,
  "fail": 0,
  "total": 560,
  "totalTime": 5076,
  "avgResponseTime": 9
}
```


Enhancements made:

- Utilized the Google Map's API to allow for an enhanced search. This improves the user experience, as the user is able to visually see all available coaches near him.
- Designed the website to look minimalistic, but neat. This improves the user experience as this makes the website easy to navigate.
- Utilized the framework bcrypt to encrypt all users' passwords when they register for an account.
- Implemented a simple booking system to allow students to book a time for coaching for a coach.
- Implemented a simple payment system (we didn't have a way to add funds because we didn't want to deal with real currency for the sake of this assignment/course).