**# Friend Finder - Node and Express Servers**

**### Overview**

In this activity, you'll build a compatibility-based "FriendFinder" application -- basically a dating app. This full-stack site will take in results from your users' surveys, then compare their answers with those from other users. The app will then display the name and picture of the user with the best overall match.

You will use Express to handle routing. Make sure you deploy your app to Heroku so other users can fill it out.

\* Check out [this demo version of the site](https://friend-finder-fsf.herokuapp.com/). Use this as a model for how we expect your assignment look and operate.

\* Create a folder called `FriendFinder`. Inside the folder, organize your directories so it matches the following:

FriendFinder

- .gitignore

- app

- data

- friends.js

- public

- home.html

- survey.html

- routing

- apiRoutes.js

- htmlRoutes.js

- node\_modules

- package.json

- server.js

**### Submission on BCS**

\* Please submit both the deployed Heroku link to your homework AND the link to the Github Repository!

**### Hosting on Heroku**

Now that we have a backend to our applications, we use Heroku for hosting. Please note that while \*\*Heroku is free\*\*, it will request credit card information if you have more than 5 applications at a time or are adding a database.

Please see [Heroku’s Account Verification Information](https://devcenter.heroku.com/articles/account-verification) for more details.

See the [Supplemental Heroku Deployment Guide](../../03-Supplemental/HerokuGuide.md) for in-detail deployment instructions

**### Instructions**

1. Your survey should have 10 questions of your choosing. Each answer should be on a scale of 1 to 5 based on how much the user agrees or disagrees with a question.

2. Your `server.js` file should require the basic npm packages we've used in class: `express` and `path`.

3. Your `htmlRoutes.js` file should include two routes:

\* A GET Route to `/survey` which should display the survey page.

\* A default, catch-all route that leads to `home.html` which displays the home page.

4. Your `apiRoutes.js` file should contain two routes:

\* A GET route with the url `/api/friends`. This will be used to display a JSON of all possible friends.

\* A POST routes `/api/friends`. This will be used to handle incoming survey results. This route will also be used to handle the compatibility logic.

5. You should save your application's data inside of `app/data/friends.js` as an array of objects. Each of these objects should roughly follow the format below.

```json

{

"name":"Ahmed",

"photo":"https://media.licdn.com/mpr/mpr/shrinknp\_400\_400/p/6/005/064/1bd/3435aa3.jpg",

"scores":[

5,

1,

4,

4,

5,

1,

2,

5,

4,

1

]

}

6. Determine the user's most compatible friend using the following as a guide:

\* Convert each user's results into a simple array of numbers (ex: `[5, 1, 4, 4, 5, 1, 2, 5, 4, 1]`).

\* With that done, compare the difference between current user's scores against those from other users, question by question. Add up the differences to calculate the `totalDifference`.

\* Example:

\* User 1: `[5, 1, 4, 4, 5, 1, 2, 5, 4, 1]`

\* User 2: `[3, 2, 6, 4, 5, 1, 2, 5, 4, 1]`

\* Total Difference: \*\*2 + 1 + 2 =\*\* \*\*\_5\_\*\*

\* Remember to use the absolute value of the differences. Put another way: no negative solutions! Your app should calculate both `5-3` and `3-5` as `2`, and so on.

\* The closest match will be the user with the least amount of difference.

7. Once you've found the current user's most compatible friend, display the result as a modal pop-up.

\* The modal should display both the name and picture of the closest match.