Note: By **field** I mean the **keys in a JSON object or a dictionary**

Below are descriptions for each route made so far:

#----------GET Routes----------#

-JSON for list/feed of mentors @app.route("/api/v1/mentors/", methods=["GET"]):
This GET request returns the following data related to the list of mentors **WHO ARE ACCEPTING CLIENTS:**

```
{
    mentors: [
        {
            "user_id" : “”,
            "name" : "",
            "phone" : "",
            "email" : "",
            "VenmoUsername" : "",
            "gender" : "",
            "height" : {
                "feet" : 0,
                "inches" : 0
            },
            "weight" : {
                "lbs" : ""
            },
            "age" : 0,
            "bio" : "",
            "tags" : [
                "strength training",
                "running"
            ],
            "role" : "",
            "location" : {
                "city" : "",
                "state" : ""
            },
            "accepting_clients" : true/false,
            "rates" : {
                "try" : 0.0,
                "loyalty" : 0.0
            },
```

```
                    "rating" : {
                            "number_of_ratings" : 0,
                            "total_score" : 0
                    },
                    "pic_url" : “”
                    partners: [ “”, “”,  … (array of partner id’s) ]
            },
            … // array of all mentors
    ],
    “url”: “”
}
```

Notes:

- The mentor feed lists the ‘partners’ as just the partner’s user_id. However, each Mentor profile (below) will display the full partner profiles instead of just their id’s
- pic_url is a single string that represents the user’s uploaded pic on S3
- **UPDATE**: only returns the Mentors who are accepting clients

-JSON for list/feed of all mentors @app.route("/api/v1/allmentors/", methods=["GET"]):
This does the exact same thing as the route above, but it returns all mentors and not just those who are accepting clients

-JSON for Mentor Profile @app.route("/api/v1/mentors/<user_id>/", methods=["GET"]):
This GET request returns the following data related to a single mentor profile:

```
{
    "user_id" : “”,
    "name" : "",
    "phone" : "",
    "email" : "",
    "VenmoUsername" : "",
    "gender" : "",
    "height" : {
            "feet" : 0,
            "inches" : 0
    },
    "weight" : {
            "lbs" : ""
    },
    "age" : 0,
    "bio" : "",
```

"tags" : [
"strength training",
"running"
],
"role" : "Mentor",
"location" : {
"city" : "",
"state" : ""
},
"accepting_clients" : true/false,
"rates" : {
"try" : 0.0,
"loyalty" : 0.0
},
"rating" : {
"number_of_ratings" : 0,
"total_score" : 0
},
"pic_url" : “”
partners: [
{
(exact same data/format as Mentee profile, but without the ‘partners’ field)
},
],
“url”: “”
}

Notes:

- Since a Mentor’s only partners with Mentees, each partner in ‘partners’ is represented as a Mentee profile but without its own ‘partners’ field to avoid recursively listing partners

-JSON for Mentee Profile @app.route("/api/v1/mentees/<user_id>/", methods=["GET"]):
This GET request returns the following data related to a single mentee profile:
{
“user_id”: “”,
"name" : "",
"role" : "Mentee",
"phone" : "",
"email" : "",
"VenmoUsername" : "",
"gender" : "",

"height" : {
"feet" : 0,
"inches" : 0
},
"weight" : {
"lbs" : 0
},
"age" : 0,
"tags" : [
"strength training",
"running"
],
"partners" : [
{
(exact same data/format as Mentor profile, but without the ‘partners’ field)
},
],
"bio" : "",
"location" : {
"city" : "",
"state" : ""
},
"pic_url" : "",
"rating" : {
"number_of_ratings" : 0,
“total_score" : 0
}
“url”: “”
}

Notes:

- Since a Mentee only partners with Mentors, each partner in ‘partners’ is represented as a Mentor profile but without its own ‘partners’ field to avoid recursively listing partners
- The difference between a Mentee and a Mentor profile is that Mentee profiles don’t have the ‘accepting_clients’ and ‘rates’ fields (aside from the ‘role’ field being different)

-JSON for Exercise Search @app.route("/api/v1/exercises/<name>/", methods=["GET"])
This GET request returns the following data related to an exercise matching the name:
{
“**exercise_id**”: “”,
name: “”,
pic_urls: [“”, “”],

“**instructions**”: “”,
“**created_by**”: “”,
“**workouts_used_in**”: 0,
“url”: “”
}

Notes:

- The name of the exercise must exactly match the name of the exercise for this to work
- **UPDATE**: the name can have a ‘+’ instead of the space, the route will filter that out

-JSON for Workout List/Search @app.route("/api/v1/<user_id>/workouts/", method=["GET"]):

This GET request returns the following data related to the workouts of a single user:

```
{
    “Workouts” :
        [
            {
                "workout_id" : “”,
                "mentor_id" : “”,
                "mentee_id" : “”,
                "workout_name" : "",
                “workout_length”: “”,
                “assigned_date”: {
                    “month”: “”,
                    “date”: “”,
                    “year”: “”,
                    “day_of_week”: “”
                }
                "exercises" : [
                        {
                            "exercise_id" : “”,
                            "exercise_name" : "",
                            "pic_urls" : [],
                            “instructions”: “”,
                            "description" : “”,
                            "notes" : "",
                        },
                        … (array of all exercises in the workout)
                    ]
            },
            … (array of workouts)
        ]
    “url”: “”
```

}

Notes:

- <id> is the unique_id of the user whose associated workouts you are retrieving. This user can be a Mentor or a Mentee. This route lists all workouts associated with this user
- Changed “exercise_key” to “exercise_id”
- Added in “instruction” and changed “specifications” to “descriptions”

-JSON for single workout @app.route("/api/v1/<user_id>/workouts/<workout_id>", method=["GET"]):

This GET request returns the following data related to a single workout involving a single user:

```
{
        "workout_id" : “”,
        "mentor_id" : “”,
        "mentee_id" : “”,
        "workout_name" : "",
        “workout_length”: “”,
        “assigned_date”: {
                “month”: “””,
                “day”: “”,
                “year”: “”,
                “day_of_week”: “”
        }
        "exercises" : [
                {
                        "exercise_id" : “”,
                        "exercise_name" : "",
                        "pic_urls" : [],
                        “instructions”: “”
                        "description" :””,
                        "notes" : ""
                },
                … (array of all exercises in the workout)
         ]
        “url”: “”
}
```

Notes:

- <user_id> is the user_id of the user whose associated workouts you are retrieving. This user can be a Mentor or a Mentee. This route lists all workouts associated with this user
- <workout_id> is the id of the specific workout you want to retrieve

-JSON for a user's requests @app.route("/api/v1/<user_id>/requests/", methods=["GET"]):

This route retrieves <user_id>'s (either a Mentor's or a Mentee's) requests. These requests can be active ("transaction_over" = False) or past ("transaction_over" = True). This GET request returns the following data:

```
{
        "requests": [
                {
                        "request_id": "",
                        "mentor_profile": (Mentor profile),
                        "mentee_profile": (Mentee profile),
                        "mentor_accepted": True/False,
                        "num_workouts_requested": 0,
                        "workouts_created": [
                                … (array of workout id's)
                        ],
                        "workouts_paid": [
                                … (array of workout id's)
                        ],
                        "transaction_over": True/False
                },
                … (array of requests)
        ]
        "url": ""
}
```

-JSON for a user's requests `@app.route("/api/v1/exercises/search/<keyphrase>", methods=["GET"])`

This route retrieves exercises based on the keyword you use. This GET request returns the following data:

```
{
   "_id" : ObjectId("..."),
   "instructions" : "...",
   "name" : "...",
   "created_by" : "<string of user id>",
   "pic_urls" : [],
   "workouts_used_in" : 0,
   "ngrams" : "... … … … … ... "
}
```

Notes:

- **A request (same as a transaction) is just a workout request that the mentee initiates towards a mentor. An open request is a request where "transaction_over" equals False. A Mentor can only create workouts if they have an open request and they have accepted that request**
- The mentor/mentee profile are exactly the same as described earlier
- "mentor_accepted" is a bool that represents whether the Mentor has accepted the Mentee's workout request
- "num_workouts_requested" is an int representing the number of workouts included in the Mentee's request
- "workouts_created" is an array of the workout id's that the mentor created for the mentee as part of this transaction/request
- "workouts_paid" is an array of the workout id's that the mentee has paid as part of this transaction
- "transaction_over" is a bool that represents whether this request has been completely satisfied (past) or is still under way (active). It is automatically set to True once 1) The mentee pays for all the workouts created; AND 2) The mentor has assigned all the "num_workouts_requested" number of workouts

-JSON for single request @app.route("/api/v1/requests/<request_id>/", methods=["GET"])
This route returns the following data about a single request specified by <request_id>:

```
{
        "request_id": "",
        "mentor_profile": (Mentor profile),
        "mentee_profile": (Mentee profile),
        "message": "",
        "mentor_accepted": True/False,
        "num_workouts_requested": 0,
        "workouts_created": [
                … (array of workout id's)
        ],
        "workouts_paid": [
                … (array of workout id's)
        ],
        "transaction_over": True/False
}
```

Notes:

- ***Update***: added in a message field

-JSON for finding the role of a user_id @app.route("/api/v1/role/<user_id>/", methods=["GET"]):

This route returns the role of a user specified by <user_id>

{

“role”: “Mentor”/”Mentee”

}

-SEND APPLICATION EMAIL

Send mentor application email (an application email will be sent to sahilg@umich.edu, application can enter whatever email address they want to sent an application to)

```
curl \
  --header 'Content-Type: application/json' \
  --request POST \
  --data '{"email":"sahilg@umich.edu"}'\
  http://104.236.13.32:5000/api/v1/sendapplication/
```

SEND VERIFICATION EMAIL (must be the user_id of a mentor, will send an email to that mentor’s id)

@app.route("/api/v1/verificationemail/<user_id>/", methods=["GET"])

@app.route("/api/v1/checkverificationcode/", methods=['POST'])

Send a JSON with fields {“verification_code”: “hello”, “email”: “a@a.com”}

Will return a JSON object with {“is_valid”: True, “user_id”: “sdfvrswef2”}

CALL JSON KEY OBJECT FOR USER DATA: user_data

CALL JSON KEY OBJECT FOR WORKOUT SEARCH: workout_search

#----------POST Routes----------#

IMPORTANT:

- For convenience, each GET request adds a “url” subfield to the data being returned. Please do not include these url’s in any PUT or POST requests as they will be ignored
- Do not include unique_id’s, exercise_key’s, or workout_id’s in POST requests, as these id’s are created and returned after the POST request is run.

Note: Each POST request’s required data is very similar in structure to their corresponding GET request’s returned data. *Exceptions (info is also available in the notes unders each route)*:

- post new user should not contain the ‘partners’ field
-

-**POST NEW EXERCISE @app.route("/api/v1/exercises/new/", methods=["POST"])**
Front end sends json data with exactly the following 4 fields:

{
"name" : "",
“pic_urls" : ["", “”, … (array of strings)],
“instructions”: “”,
“created_by”: “” (string of mentor id who created this exercise)
}

The backend returns the newly formed exercise if
{
“**exercise_id**”: “”
}

-POST NEW REQUEST @app.route("/api/v1/requests/new/", methods=["POST"]):
Front end needs to send json data with exactly the following 3 fields:
{
“mentee_id”: “”,
“mentor_id”: “”,
“num_workouts_requested”: “”,
“message”: “”
}

The backend returns the newly formed request if

{

“request_id”: “”

}

Notes:

- This route automatically creates “workouts_created”, “workouts_paid” and sets them to empty arrays. It also automatically created “mentor_accepted”, “transaction_over” and sets them to False.
- ***Important***: You cannot create a new request if there is already an open request (“transaction_over” = False) between the same pair of Mentor & Mentee. All requests must be satisfied before new requests can be created for the same pair.
- ***Update***: added in a message field

-POST NEW WORKOUTS @app.route("/api/v1/workouts/new/", methods=["POST"])

Front end needs to send json data with exactly the following 6 fields (and 5 subfields inside the ‘exercises’ field):

{

"mentor_id" : "",

"mentee_id" : "",

"workout_name" : "",

“workout_length”: “”,

“assigned_date”: {

“month”: “””,

“day”: “”,

“year”: “”,

“day_of_week”: “”

}

"exercises" : [

{

"**exercise_id**" : "",

"exercise_name" : "",

"pic_urls" : [“”, “”, …],

“**instruction**”: “”,

“description”: “”,

"notes" : ""

},

… (array of all exercises in the workout)

]

}

The backend then returns the newly formed workout_id:

{

“workout_id”: “”

}

Notes:

- When the front end searches for an exercise to add to the workout, the exercise details (the 5 subfields in the ‘exercises’ field as shown above) are not automatically added in by this route. The front end must manually enter these details. The format shown above is exactly how the database stores workouts (compared to, for example, the get-mentor/mentee routes that automatically return the partners’ profiles while the database just stores the partners’ id’s). Just wanted to clarify.
- ***Important***:

  1) Calling this route automatically increments each included exercise’s “workouts_used_in” count by 1. *BUG:* this increment function doesn’t increment correctly, need to fix it.

  2) A workout can only be created by a Mentor if they have an open transaction AND they haven’t already created the requested number of workouts already

  3) If a workout is successfully created, it is added to the “workouts_created” array of the associated request (remember that a Mentor-Mentee pair can only have one open request/transaction)

~~**POST/Add new Partners @app.route("/api/v1/users/addpartner/", methods=["POST"]):**~~

UPDATE: this route has been removed since it is taken care of when accepting a request

~~Front end needs to send json data with exactly the following 2 fields:~~

~~{~~

~~“mentor_id”: “”,~~

~~“mentee_id”: “”,~~

~~}~~

~~The backend returns the following data:~~

~~{~~

~~“**mentor_id**”: “”,~~

~~“**mentee_id**”: “”,~~

~~}~~

~~Notes:~~

- ~~This route adds <mentee_id> to the mentor’s ‘partners’ field and vice verse~~

- ~~Changed “mentor” and “mentee” in the backend returned data to “mentor_id” and “mentee_id”~~

-POST/Add new user @app.route("/api/v1/users/new/", methods=["POST"]):
Front end needs to send json data with exactly the following 16 fields if Mentee or 17 fields if Mentor.

Required data for **new Mentee:**

```
{
        “user_id”: “”,
        "name" : "",
        "role" : "Mentee",
        "phone" : "",
        "email" : "",
        "VenmoUsername" : "",
        "gender" : "",
        "height" : {
                "feet" : 0,
                "inches" : 0
        },
        "weight" : {
                "lbs" : 0
        },
        "age" : 0,
        "tags" : [
                "strength training",
                "running"
        ],
        "bio" : "",
        "location" : {
                "city" : "",
                "state" : ""
        },
        "pic_url" : "",
        "rating" : {
                "number_of_ratings" : 0,
                “total_score" : 0
        },
        “partners”: []
}
```

Required data for **new Mentor:**

```
{
        "name" : "",
        "phone" : "",
        "email" : "",
        "VenmoUsername" : "",
        "gender" : "",
        "height" : {
                "feet" : 0,
                "inches" : 0
        },
        "weight" : {
                "lbs" : ""
        },
        "age" : 0,
        "bio" : "",
        "tags" : [
                "strength training",
                "running"
        ],
        "role" : "Mentor",
        "location" : {
                "city" : "",
                "state" : ""
        },
        "accepting_clients" : true/false,
        "rates" : {
                "try" : 0.0,
                "loyalty" : 0.0
        },
        "pic_url" : “”,
        "rating" : {
                "number_of_ratings" : 0,
                "total_score" : 0
        },
        “partners”: []
}
```

The backend returns the newly created user_id (same as unique_id, I need to go through the code to converge on a single name for these id’s):

```
{
        “user_id”: “”,
```

“mongo_id”: “”
}

If you specified a “user_id” for a new Mentor, a warning message is added to the returned data:
{
“user_id”: **“”** (empty string, not a string placeholder like usual),
“mongo_id”: “”,
“warning_message”: “user_id was provided but will be set to an empty string (new Mentor)”
}

Notes:

- If the user is a new Mentor, data must have the 2 extra fields: “accepting_clients” and “rates”.
- **If the new user is a Mentor and data has a “user_id” field, it will be re-written to an empty string (even if data doesn’t have “user_if”, it will be set to an empty string when written to the database).**
- **Mentees must have the “user_id” field though.**
- The “partners” field for both a new Mentor and a new Mentee **must be an empty** array or an exception will be thrown
- The “rating” field for both a new Mentor and a new Mentee **must have “total_score” = 0 and “number_of_ratings” = 0** or an exception will be thrown

-POST/Update a Mentor’s user_id field @app.route("/api/v1/mentors/setuserid/", methods=["POST"])
This route should be called after a Mentor has been created and a firebase token has been generated for it. Front end sends data with exactly the following 2 fields:
{
“mongo_id”: “”,
“user_id”: “”
}
Backend returns the following data:
{
“user_id”: “”
}

Notes:

- This route only works if 1) “mongo_id” specified a Mentor; AND 2) The mentor’s “user_id” field in the database is currently an empty string (default when creating a new mentor)

- “user_id” in data should be the firebase token that is desired to be stored as “user_id” in the database

#----------PUT Routes----------#

***Important*:** Frontend needs to make sure that when something is being edited, the current values from the database are populated/shown to the app’s user by default. Then, the app’s user can edit what they need. This is because each PUT request replaces the entire document with the values specified in the PUT request’s data, not just the edited values.

-EDIT USER STRUCTURE @app.route("/api/v1/users/edit/<user_id>", methods=["PUT"]): Frontend needs to send data that has the same structure as described previously **EXCEPT the ‘partners’ field**. So, this route’s data should have 14 fields for editing a Mentee and 16 fields for editing a Mentor. However, data **must contain the ‘role’ field although it also cannot be edited**. Below are the updates that take place in the route. Below is how the fields for a Mentee are updated in our code using the passed-in data. For a Mentor, you can also update “accepting_clients” and “rates”:

```
 "name":data["name"],

"phone": data["phone"],
"email": data["email"],
"height": data["height"],`
"weight": data["weight"],
"age": data["age"],
"bio": data["bio"],
"tags": data["tags"],
"location": data["location"],
“pic_url”: “pic_url”,
“VenmoUsername”: data[“VenmoUsername”],
“gender”: data[“gender”]
```

Notes:

- You can update “accepting_clients” and “rates” in the same fashion as above only if you specify data[“role”] = “Mentor”

- You **cannot update** a user's "role" field (which represents whether they are a Mentor or a Mentee).
- **EDIT**: you **cannot update** a user's "partners" in this method for simplicity. Adding a partner can be done with the add-partner POST request. Partners cannot be removed for now, I can look into changing that later

~~**-EDIT EXERCISE @app.route("/api/v1/exercises/edit/<id>/", methods=["PUT"]):**~~
~~Similarly, the exercise route needs exactly the 2 fields from the exercises structure. Below is how our code updates an exercise specified by <id> using the passed-in data:~~

~~{~~
~~"name": data["name"],~~
~~"pic_urls": data["pic_urls"],~~

~~}~~
UPDATE: talked to Levi, we don't need this route anymore

-ACCEPT REQUEST
@app.route("/api/v1/users/<mentor_id>/acceptrequest/<request_id>/", methods=["PUT"]):
This route should be called when a Mentor receives a new request and they hit "accept".
Front end doesn't need to send data (aside from the <mentor_id> and <request_id> in the API url)
Backend as of now just returns a 200 code.

Notes:
- This method sets the "mentor_accepted" field to True in the request specified by <request_id>
- If mentor_id and mentee_id (mentee_id obtained from request_id automatically) aren't already partners, this route adds them to eachother's "partners" array in their user profile

-DENY REQUEST @app.route("/api/v1/users/<mentor_id>/denyrequest/<request_id>/", methods=["PUT"]):
This route should be called when a Mentor receives a new request and they hit "deny".
Front
Front end doesn't need to send data (aside from the <mentor_id> and <request_id> in the API url)
Backend as of now just returns a 200 code.

Notes:

- This route deletes the request specified by <request_id> from the database **(The database only contains the requests that are yet to be accepted or have already been accepted)**
- You cannot deny a request if it has already been accepted

-PAY FOR A WORKOUT ROUTE
@app.route("/api/v1/users/<mentee_id>/paid/<workout_id>/", methods=["PUT"]):
This route should be called when a Mentee pays for a workout assigned by the Mentor with whom they have an open request/transaction.
Front end doesn't need to send data (aside from the <mentee_id> and <workout_id> in the API url)
Backend as of now just returns a 200 code.

Notes:
- A Mentee can only pay for a workout if 1) they have an open transaction with a Mentor; AND 2) there are workouts in the "workouts_created" array of the request that haven't been paid yet
- If this route is called on the last workout that the Mentee has left to pay in the transaction, the request's "transaction_over" is automatically set to True since the transaction will be satisfied. (This happens if, after the payment is over, the number of workouts in "workouts_paid"/"workouts_created" is equal to "num_workouts_requested"

-EDIT WORKOUT @app.route("/api/v1/workouts/edit/<id>/", methods=["PUT"]):
Update: Front end doesn't need this route, but still keeping it up incase.
The workout edit route needs exactly the 5 fields from the workouts structure. Below is how our code updates a workout specified by <id> using the passed-in data:

```
{
        "workout_name": data["workout_name"],
        "workout_length": data["workout_length"],
        "assigned_date": data["assigned_date"],
        "exercises": data["exercises"]
}
```

Notes:
- As with the POST route for a new workout, the front end needs to make sure each exercise in the 'exercise' field has the 5 valid subfields. I will add sanitizing code later though.
- Added in "assigned_date"

- **Update**: Changed the route so you can't edit the mentor_id or mentee_id. This is because of the decision to include the "workout requests" system, so a Mentor can only create workouts for the user who has sent them a request.