Hector Alvarez  
FallB Oct 27th 2021  
Week2 Capstone

Service Layer Design:

For this application’s backend, I will use NodeJS and Express to create the application RestAPI.

For this route, the user will post to a form on the front end, which will send an object to this endpoint. The data for this field will include an user first name, last name, password, email and age. This data will be processed by the mongoose schema in this endpoint, and if successful, will return a status 200 code. Otherwise, it will send back an error code that the account could not be made. Error messages will be handled on the front end, and will display a pop up that the sign up was not successful and to try again. For example if the email is already in use, the pop up will contain that information.

app.post('/signup', async function(req, res ) …

This route will be for an existing account, the front end will pass an object from a form containing fields : email and password. The endpoint will access the database, and will check for fields {email } to return user if it exists. The email will be hashed at this point and will have to be rehashed. A comparator of the email can be created, and if both email and passwords match, the user can gain access to their account.

app.post("/login", async (req, res) => { …

once the user has logged in, their unique object ID (from the user schema) will be their dynamic profile route. Users can then user this route to add a new weight data point to their account. As shown in the database design, the application will have a field that includes: new weight , units , and will automatically add a timestamp. The database will save their unique id as reference that it belongs to this user. Since the only information that is required by the user is a weight point, and all types are controlled by the json in the front end, no errors should happen at this stage ( negative numbers will not be an input option).

app.post("/profile/:id/addweight" , async (req, res) => {

Similarly, the user will be able to add new workout data to their profile. This data will include the muscle group they worked out, the load, the units, sets and repetition. This data will be important for analytics down the road.

app.post("/profile/:id/addworkout", (req, res) => {

When a user wants to see a chart/graph of their progress, the application will use this endpoint to find all the data points associated with the user (found by the user schema unique id referenced). From all the users data points, we can plot them according to a time period.

app.get("/profile/:id/weighthistory", (req, res) => {

Similarly, the user will be able to check the progress on their workouts. By filtering the data by body region worked out, we can graph each data accordingly to show progress by region worked out, or by exercise done.

app.get("/profile/:id/workouthistory", (req, res) => {

Using (these) endpoints (changeemail, changepassword etc) we can edit the users current data and allow them to change both those fields.

app.patch("/profile/:id/edit#" , async (req, res) => {..

using this endpoint, the user will be able to access certain videos for the workout that they require. The dynamic :id route will allow the user to either look up :id = arms, or legs , bicep , chest, back videos etc. This will allow the data to be organized, and be able to be found by name. more than one video should be returned.

app.get("/trainercontent/:id, (req, res) => {..

SERVICE LAYER DIAGRAM

Diagram

Description automatically generated