Back-End Application Programming Interface (API)

The Strength Research Online application front-end exchanges and updates data stored in the PostgreSQL database tables using this Application Programming Interface (API). The Express back end provides the APIs to the front-end.

Each API iincludes the name of a method in the front-end code that uses this API. The baseURL function resolves the URL of the back-end server to create a call that can be used us HTTP gets and posts.

Running the Back-End on the web server

The APIs are functions stored inside the Node.js program **Strength\_Coaching\_Back\_End.js**. This runs continuously on the server, started automatically by the **pm2** process manager each time the server is restarted:

pm2 start node --env-file=sr.env Strength\_Coaching\_Back\_End.js

APIs

/api/authenticateUser

Authenticates the user by verifying that their registration user\_ID number, alias, or their email address can be located in the database. If found, the API checks that the password supplied matches the encrypted one stored in the database.

Within the back-end, the **bcrypt.compare()** function is used to compare the encrypted password retrieved from their record with the password supplied. If the user password matches, a JSON Web Token (JWT) is generated and returned to the client. Refer to the back-end code for the current bcrypt Salt constant, the JWT Secret, and the JWT Expiry Time.

const authenticateUser = async (UserID, Password) => {  
 try {  
 let response = await axios.get(baseURL + "authenticateUser?user\_ID=" +  
 encodeURIComponent(UserID) +  
 "&password=" + encodeURIComponent(Password)  
 );

if (response.status === 200) {  
 // The user was found and their credentials were authenticated.

/api/createUser

Creates a new user with only their first name, last name, their email address, and their registration token in the new record. The user\_ID is an auto-incrementing unique key that is also returned in the response packet. Note that this is an async function since the database needs time to retrieve the record and return a promise.

const createUser = async () => {

/api/getToken

This function generates and returns a JSON Web Token (JWT) that contains a hashed user ID and an expiry time specified by the calling function. It uses the same JWT secret as the other token-generating functions in this back-end. When re-submitted later, it can be verified using the **verifyToken()** function to ensure it has not timed out and that the same user is accessing the system during this session. The token is returned in **response.data.token.**

try {  
 let response = await axios.get(baseURL + "getToken?user\_ID=" + user\_ID +   
 "@expiry\_time=" + expiry\_time);  
 if (response.status === 200) {   
 console.log("Token = " + response.data.token);  
 }  
}

This API is used in **ResetPassword.jsx** to create a token that verifies the user when they access the Reset Password page via a URL link sent to them in an email.

/api/getUser

This API returns an individual user record based on their user\_ID. This api must receive a valid JWT for the current session before it executes. Example calls are found in **EditMyProfile.jsx**. Refer to the **authenticateUser** API for retrieving information during sign-in and other processes where not JWT has been assigned.

const getUser = async (userID) => {   
 try {  
 let response = await axios.get(baseURL + "getUser?user\_ID=" + userID + "&JWT=" + JWT);  
 if (response.status === 200) {   
 setUserID(response.data.user\_ID);  
 }  
 }  
}

/api/createUser

This API API to create a new user with only their first name, last name, their email address, and their registration token in the new record. The user\_ID is an auto-incrementing unique key that is returned in the response packet. This example is from the **Registration.jsx** page:

const createUser = async () => {  
 var status = false;

axios.put(baseURL + "createUser", {  
 user\_authority: "U",  
 password: Password,  
 user\_status: "A",  
 registration\_token: "",   
 verification\_code: VerificationCode,   
 first\_name: FirstName,  
 last\_name: LastName,  
 email\_address: EmailAddress,   
 user\_image: default\_user\_image  
 })  
 .then((response) => {  
 if (response.status === 200) {   
 status = true;   
 }  
}