Strength Research Online Database v3.2

Table: User

The User table holds information about each individual user. Users can be administrators, trainers, or clients. There is a natural hierarchy of trainers who work with multiple clients and clients who employ different trainers for different disciplines. The application provides functionality that allows trainers to bring on-board new clients, manage their coaching programs, and transfer them to other associated trainers.

Verification rules are applied in the React maintenance application in the front-end ensure that only valid combinations of these attributes are allowed to be specified for each user.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | | ***Description*** | ***Data Type*** | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| user\_ID | | Unique six-digit code to identify each user. A user can be an administrators, trainers, or clients. This ID is automatically assigned when the user record is created and can be used to log-in to the site. | VARCHAR(6) | N6 | 1 to 999999 | Y | PK |
| user\_authority | | The authority level of this user:   |  |  | | --- | --- | | **S** | A super-user or administrator who has access to all functionality. Administrators can bring on-board new coaches as well as being coaches themselves.. | | **C** | Coaches manage the training plans for clients. They have full access to the information of all the clients that they manage. | | **U** | A User, who is a Client of a Coach. They can manage their own identity and contact information, but cannot create their own training plans. | | **A** | AI who can use a restricted set of Back-End APIs to access anonymised client information to assist a coach. | | CHAR(1) |  |  | Y |  |
| password | | Case-sensitive password. The stored password is hash-encrypted when it is created or changed with a one-way **bcrypt()** salted hashing algorithm.  During password verification, the bcrypt library provides a function to verify the plain-text password entered against the encrypted one. | VARCHAR(70) | 60 |  | Y |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | | ***Description*** | ***Data Type*** | ***Format*** | ***Range*** | ***Req*** | ***Key*** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| user\_status | Tracks the current status of the user and their password:   |  |  | | --- | --- | | **A** | The user has a valid, active password and can access the site. | | **R** | The user is currently in the process of  resetting their password, possibly because they forgot it. | | **N** | The used is new and in the process of registering. | | | CHAR(1) |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| registration \_token | The registration activation or password reset token sent to the user by email. | | VARCHAR(500) |  |  |  |  |
| verification\_code | The numeric verification code the client needs to enter when resetting their password or registering a new account | | VARCHAR(10) |  |  |  |  |
| token\_date\_time | The date and time the registration or reset token was created. This is used during the password reset process to invalidate expired tokens. The current token lifetime is set to 15 minutes | | DATETIME |  |  |  |  |
| last\_sign\_in\_date | Date and time of the registrant’s last sign in to the system. | | DATETIME |  |  |  |  |

Name and contact information

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | ***Description*** | | ***Data Type*** | | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| salutation | Salutation including Mr, Mrs, Sir, Dr, Lady, Lord, and Rev. | | VARCHAR(10) | |  |  |  |  |
| first\_name | First name | | VARCHAR(30) | |  |  | Y |  |
| middle\_name | Middle name or initial | | VARCHAR(30) | |  |  |  |  |
| last\_name | Last name or surname | | VARCHAR(50) | |  |  | Y |  |
| alias | Short alias name for the user (e.g. LukeS). It must be unique because it can be used to log in with the password instead of entering the user\_ID or email\_address. It is checked for uniqueness during maintenance before being accepted. | | VARCHAR(30) | |  |  |  |  |
| gender | Gender. Not currently used but reserved for future use. Valid options include:   |  |  | | --- | --- | | **M** | Male. | | **F** | Female. | | **O** | Other. | | **N** | Not specified. | | | CHAR(1) | |  |  |  |  |
| ***Attribute Name*** | ***Description*** | | ***Data Type*** | | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| pronouns | Pronouns for specifying gender including him/he, she/her. This is a free-form field that is not validated. Not currently used but reserved for future use. | | VARCHAR(20) | |  |  |  |  |
| phone\_number | Mobile phone number or landline number. Maximum length is based on the ITU-T E164 standard recommendation available from here: <https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=E.164> | | VARCHAR(15) | |  |  | Y |  |
| email\_address | Email address. Validated using the RFC 5322 standard such that:   * The local part is no more than 64 characters long. * The @ sign is present. * The domain part is no longer than 255 characters.   <https://datatracker.ietf.org/doc/html/rfc5322>  The email address can also be used to sign-in since it is unique on the system. It is validated at sign-on time. | | VARCHAR(320) | | #@#.# |  | Y |  |
| address\_1 | Address line one. | | VARCHAR(30) | |  |  | Y |  |
| address\_2 | Address line two. | | VARCHAR(30) | |  |  |  |  |
| address\_3 | Address line three. | VARCHAR(30) | |  | |  |  |  |
| suburb | Suburb. | VARCHAR(30) | |  | |  | Y |  |
| city | City name. | VARCHAR(30) | |  | |  | Y |  |
| postcode | Postcode. | VARCHAR(30) | |  | |  | Y |  |
| state\_province | State or province. | VARCHAR(30) | |  | |  |  |  |
| country | Country name | VARCHAR(30) | |  | |  | Y |  |
| date\_of\_birth | Date of birth | DATE | |  | |  | Y |  |
| gym\_name | Name of the gym the user is registered at or the trainer operates out of. This could be a foreign key to a list of gyms the trainers are associated with. | VARCHAR(30) | |  | |  |  | FK |
| user\_image | The file name of the image the user has uploaded to the sites Public userImages directory on the server. This image is displayed on the top of the page after they authenticate and in other places like their Block training schedule. | VARCHAR(50) | |  | |  |  |  |

Payment plan information

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | ***Description*** | ***Data Type*** | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| payment\_type | The current payment plan of this user: | CHAR(2) |  |  | Y | FK |

Emergency contact information

This section carries the details of the assigned guardian of a juvenile dancer. If the calculated registrant age is less than or equal to seven, then guardian information must be supplied. Reserved for future use and not implemented at the moment.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | ***Description*** | ***Data Type*** | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| emg\_contact\_name | The name of the person who is to be contacted in the event of an emergency. | VARCHAR(60) |  |  | Y |  |
| emg\_contact \_phone | The emergency contact phone number | VARCHAR(15) |  |  | Y |  |
| emg\_contact email\_address | The email address of the contact. | VARCHAR(60) |  | #@#.# | Y |  |

Table: Configuration

Holds the configuration settings used by the application.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | ***Description*** | ***Data Type*** | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| config\_ID | The key to the configuration file. This is always 1. | SMALLINT | N | 1 to 1 | Y | PK |
|  | *More to come ….* |  |  |  |  |  |

Table: Training plan

These are a set of linked tables that contain the training plans for an individual client. This data can be planned for future work, current plans being worked on by the client, or historical completed records.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Attribute Name*** | ***Description*** | ***Data Type*** | ***Format*** | ***Range*** | ***Req*** | ***Key*** |
| plan\_ID | The key to the client this plan is for. | TBA |  |  | Y | PK |
|  | *More to come ….* |  |  |  |  |  |

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.

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.

/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;   
 }  
}