Calorie Tracker Technical Manual

tracker.aasifversi.com

James Childress
Matthew Goodman
Enrique Mitchell
Aasifali Versi

Stack:
Calorie Tracker is a Node.js web application hosted on an AWS EC2 instance using a MySQL server to store user information. The application itself uses Express as the server-side routing layer, Embedded JavaScript (*.ejs) templates compiled to HTML for views, and vanilla JavaScript for all additional scripts.

Database Access and Database Schema:

The MySQL database is personally hosted by Matthew Goodman through his AWS account. Access to this database will be terminated after the course. To access the database either login through the AWS console (credentials will not be provided) or use the MySQL command-line interface (shown below). ***Note***: to use the MySQL command-line interface a version of MySQL must be downloaded. Any version of the MySQL Community Server package should suffice as long as command-line utilities are included.

```
mysql -h cis350group10db.cca9bnl1w9fn.us-west-2.rds.amazonaws.com -P 3306 -u
cis350_group10 -p
```

The password is: *helaBroUiZAL*

The server contains five tables: ***FOOD_ENTRIES***, ***exercise***, ***favorites***, ***users***, and ***weight***. All table names are case-sensitive. The schema for each table follows:

To briefly describe MySQL table schema, every entry in a MySQL table is a “row”. The fields of the entry are specified as columns. Every MySQL field has specific parameters associated with it. These parameters compose the “schema” that we use to describe a table. The schema associated with each table can be found below. The basic parameters that describe any field are its name (“Field”), data type, whether or not it accepts a null value (“Null”), and default value. Two other important parameters are properties of the key (“Key”) and extra options (“Extra”). The two descriptors we will see in the “Key” parameter are “MULTIPLE” and “PRIMARY”. The value “PRIMARY” specifies that this field is unique to all entries in the table and can therefore be used to distinguish them. The value “MULTIPLE” specifies that this field can have the same value across multiple entries. In the FOOD_ENTRIES schema below where every row is a diary entry, a user can submit multiple diary entries, therefore the field is MULTIPLE.

Schema: **FOOD_ENTRIES**
Description: Global store of all diary entries from all users

Field	Type	Null	Key	Default	Extra	Description
Entry_ID	int(11)	NO	PRIMARY	NULL	auto_increment	Unique ID number for each diary entry
Entry_date	char(10)	NO		NULL		Date of diary entry
username	char(100)	NO	MULTIPLE	NULL		User that made diary entry
Entry_Content	text	NO		NULL		Content of diary entry

Schema: **exercise**
Description: Global store of all exercise entries from all users

Field	Type	Null	Key	Default	Extra	Description
Entry_Date	char(10)	YES		NULL		Date of exercise entry
username	char(100)	YES		NULL		User that made exercise entry
Entry_Content	text	YES		NULL		Content of exercise entry

Schema: **favorites**
Description: Global store of all favorites entries for all users. User can add diary entry to favorites so they can reuse it later.

Field	Type	Null	Key	Default	Extra	Description
fav_ID	int(11)	NO	PRIMARY	NULL	auto-increment	Unique ID of favorite entry
name	char(100)	NO		NULL		Name of favorite entry
carbs	int(11)	YES		NULL		Carbs of favorite entry
fat	int(11)	YES		NULL		Fat of favorite entry
protein	int(11)	YES		NULL		Protein of favorite entry
username	char(100)	NO	MULTIPLE	NULL		User that made favorite entry
unit	char(100)	YES		NULL		Measurement unit of favorite entry
serving	int(11)	YES		NULL		Num. servings of favorite entry
meal	char(100)	YES		NULL		Meal of favorite entry (lunch, dinner, etc.)

Schema: **users**
Description: Stores user’s usernames, password, real information, phone number, and goals. Note that only hashes of passwords are stored

Field	Type	Null	Key	Default	Extra	Description
username	char(100)	NO	PRIMARY	NULL		Username
firstname	char(100)	NO		NULL		First name of user
lastname	char(100)	NO		NULL		Last name of user
password	char(200)	NO		NULL		Hash of user password
calories	int(11)	YES		NULL		Calorie goal of user
carbs	int(11)	YES		NULL		Carbs goal of user
fat	int(11)	YES		NULL		Fat goal of user
protein	int(11)	YES		NULL		Protein goal of user
goals	tinyint(1)	YES		NULL		User has specified goals (boolean)
food	tinyint(1)	YES		NULL		User has specified food (boolean)
phoneNumber	char(200)	YES		NULL		User phone number

Schema: **weight**
Description: Global store of all weight entries from all users

Field	Type	Null	Key	Default	Extra	Description
Entry_Date	char(10)	YES		NULL		Date of weight entry
username	char(100)	YES		NULL		User that made weight entry
userWeight	int(11)	YES		NULL		Numeric value weight entry

Views and Required Components: screenshots of views are in user manual.

View: **login.ejs**
Route shown when user is asked to login. This gives you the option to sign in with your credentials or with facebook login. **Note:** facebook login only works at https://tracker.aasifversi.com

View: **users.ejs**
Route shown when user logs in. Presents options to add new food entries, new exercise entries, etc. Also shows entries for the day and presents option to show entries from previous days. Connects directly to **users.js** to directly handle page requests.

View: **public.ejs**
Shown when someone tries to access a public page of a user. Connects directly to **public.js** to directly handle page requests.

View: **favorites.ejs**
Shown when someone tries to access his or her own favorites list. Connects directly to **favorites.js** to directly handle page requests.

View: **signup.ejs**
Route shown when user proceeds to signup. Presents option to either sign up using form or with facebook. Connects directly to **signup.js** to handle page requests. This includes the routing to handle a Facebook signup attempt. When this occurs, the request produces an AJAX request that is handled by **resolveLoginCredentialsFacebook.js** in the *public/javascripts* folder.

View: **newEntry.ejs**
Route shown when user attempts to make manual food entry (displayed as button in **users.ejs** and other logged in pages). Connects directly to **newEntry.js** to handle page requests. Validation is performed both client-side and serverside. Client-side validation is handled using a script embedded in the view code that imports **parsley.js** (found in *public/javascripts*). Likewise, **newEntry.js** handles server-side validation.

View: **search.ejs**
Route shown when user attempts to make search food entry (displayed as button in **users.ejs** and other logged in pages). When the user enters a search term, they are making a request to the node back end which then uses the nutritionix API

Backend: **routes/*.js**
Each ejs file has a corresponding js file. This js file is used to serve the data on the ejs page as described above.

Backend: **database/databse.js**
This file takes care of establishing the connection to our SQL database all other files use this to access the database.

Credentials:
All the necessary credentials are stored in the file **credentials.json**. These credentials are used to access the apis that we are using and our database.

Config:
The file config.json contains the houndify api keys and the configuration required to make voice recognition work on local and remote

How to run our server locally:
Using the zip file we have provided there are a few steps necessary to get the project running.

- Install nodejs
- Open a terminal in the directory where you have unzipped the project
- Run ‘npm install -g gulp’, you may need to use sudo privileges.
- Run ‘npm install’
- Run ‘gulp’
- You can now access the site at localhost:3000

Alternative method to run website on server - Docker:

- Open a terminal. Run command below.
- `sudo docker run --name website -p 3000:80 goodmanm/calorieTracker:v3`