Milestone # 4 - Database Design File
Team Name: Midsummer Night's Team

Members: Justin Adkins, Jamison McGinley, Jennifer Gurtler, Julien Rumsey, Shania Roy, Parikshit Bhetwal

Database Software, and Database Creation Scripts:

We chose to use PostgreSQL for our DBMS, utilizing PgAdmin3/4 as a tool to visualize the data as well as to query our data once it was seeded. We have two tables in our database. One that holds user information for registering/authentication and another for data created by the user on a session to session basis. This second table will be relatively volatile, and the data added to it will be deleted if the user signs out or selects to remove it from the site. For all our queries done within our code we have used a JavaScript query builder utility called Knex.js. Knex takes PostgresSQL calls and translates them into functional JavaScript that we can use in our code. One of the utilities Knex provides is the ability to migrate tables into our database and "seed" them with data provided in a JavaScript file. The code below is a snippet from one of our migration files to give you an idea of how it works.

```
exports.up = function(knex, Promise) {
  return knex.schema.createTable('users', function(table) {
    table.increments('user_id').primary();
    table.text('email').unique().notNullable();
    table.text('password').notNullable();
    table.json('user_courses');
    table.datetime('created_at').notNullable();
  });
};
```

Once this code is implemented, we simply execute the command:

```
knex migrate:latest
```

This will build the table in our connected database (if it doesn't already exist). The PostgresSQL query translation, for reference, of the above would be:

```
CREATE TABLE public.users
(
    user_id integer NOT NULL DEFAULT nextval('users_user_id_seq'::regclass),
    email text COLLATE pg_catalog."default" NOT NULL,
    password text COLLATE pg_catalog."default" NOT NULL,
    user_courses json,
    created_at timestamp with time zone NOT NULL,
    CONSTRAINT users_pkey PRIMARY KEY (user_id),
    CONSTRAINT users_email_unique UNIQUE (email)

)
```

To seed our table with data, we then use a separate file that contains the following:

```
const bcrypt = require('bcrypt');
import { users } from './userData';

exports.seed = function(knex, Promise) {
  // Deletes ALL existing entries
  return knex('users').truncate()
    .then(function () {
            return knex('users').insert(users)
    })
};
```

With how this code is currently written, it allows us to wipe the *entire* database and re-seed with ease. A "users" object is imported which is an array of JSON objects that contain user data. We then execute the command to send the queries to the database which will pull from all our seed files:

```
knex seed:run
```

We are not using SQL scripts to populate our database because our seed files allow us to do so. You can find these files from the path CSCI_3308Project/serverSide/db/seedData/ on our GitHub. You will find **userData.js and postData.js** within the path specified above. userData.js will have all the data necessary to populate the "Users" table for testing and postData.js will have all the data necessary to populate the "Posts" table for testing.

The following is a snippet from **userData.js** to give a better understanding of this:

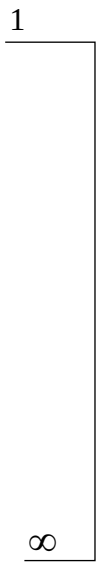
```
const bcrypt = require('bcrypt');

module.exports = [
  {
    email: 'foobar@gmail.com',
    password: bcrypt.hashSync('pineapple',10),
    user_courses: {
      course_1: "CSCI_1300",
      course_2: "CSCI_2270"
    },
    created_at: new Date()
  },
  {
    email: 'hello@gmail.com',
    password: bcrypt.hashSync('frogger', 10),
    user_courses: {
      course_1: "CSCI_2270",
      course_2: "CSCI_2400"
    },
    created_at: new Date()
  },
```

We have a very similar migration and seed file for our second table and to avoid redundancy, I'll leave it to the reader to investigate our GitHub if they'd like to see the code.

Data Model:

Attributes	Users
PK, Int	**User_ID**
Text	Email
Text	Password
JSON	User_Courses
Timestamp	Created_At

Attributes	Posts
PK, Int	**Post_ID**
FK, Int	User_ID
JSON	Courses
Text	Location
VARCHAR(140)	Note



The above data model depicts our two tables and their relation to one another. They are linked by a User_ID primary key in the first table with a User_ID foreign key in the second. The diagram shows how one “User” can have multiple posts, but each post must belong to only one user. Our attribute types are listed on the left.