-SCENARIO-

From the Git Classroom link, you will be provided with a web application for Animal adoptions, which uses static data – the data is currently stored in JSON format directly in the project code.
The web application is currently not fully functional.
The Front-end of this web application is already set up. You may update anything in the Front-end ONLY if it is required to make your back-end functional.

After signing into the web application, users can adopt one or more animals from a provided list of animals.
The application must be changed to accommodate a MySQL database to store animals', logged-in users' and adoption data.

Users can navigate the application using a navbar, as provided in the web application. There are five tabs in this navbar, linking to the following pages:

- Home page – landing page.
- Animals page – lists all animals in the database, regardless of adoption status. (Only logged-in users with the 'member' role can adopt animals).
- Species page – only accessible by the 'admin' user role, species information can be updated here.
- Temperament page – only accessible by the 'admin' user role, temperament information can be updated here.
- Sign in page – where users will be able to sign in, and sign up.

Your tasks in this course assignment are as follows:

- Change the back-end for this web application to use a MySQL database.
- Create the relevant tables, columns, and data types with the relevant relationships between the tables, in the 3rd normal form, using the provided data from the table on the Animals page.
- The application must authenticate users with their username and password in a 'users' table.
- Use Sequelize to handle the connection between the client and the database.
- Use Sequelize to create the database structure.
- Use Sequelize raw queries to insert the initial row data, which must be read from queries stored in JSON files.
- All queries (CRUD) must be executed with Sequelize, except creating/adding the initial data in the database (Unless otherwise specified).

-SETUP WORKSPACE-

Code:

Download/clone the pre-created web app from the Git link shared in the class channel.

Install the necessary dependencies for the web application to run correctly if any are needed.

Readme file

In the readme file of your GIT hub project (Use the README file provided in the .zip folder), include the following:

- A detailed description of how to use the application (installation instructions, how to run the application, etc)
- Information on the environment variables that are needed
- Any additional external libraries/packages that were used
- Which NodeJS version is being used

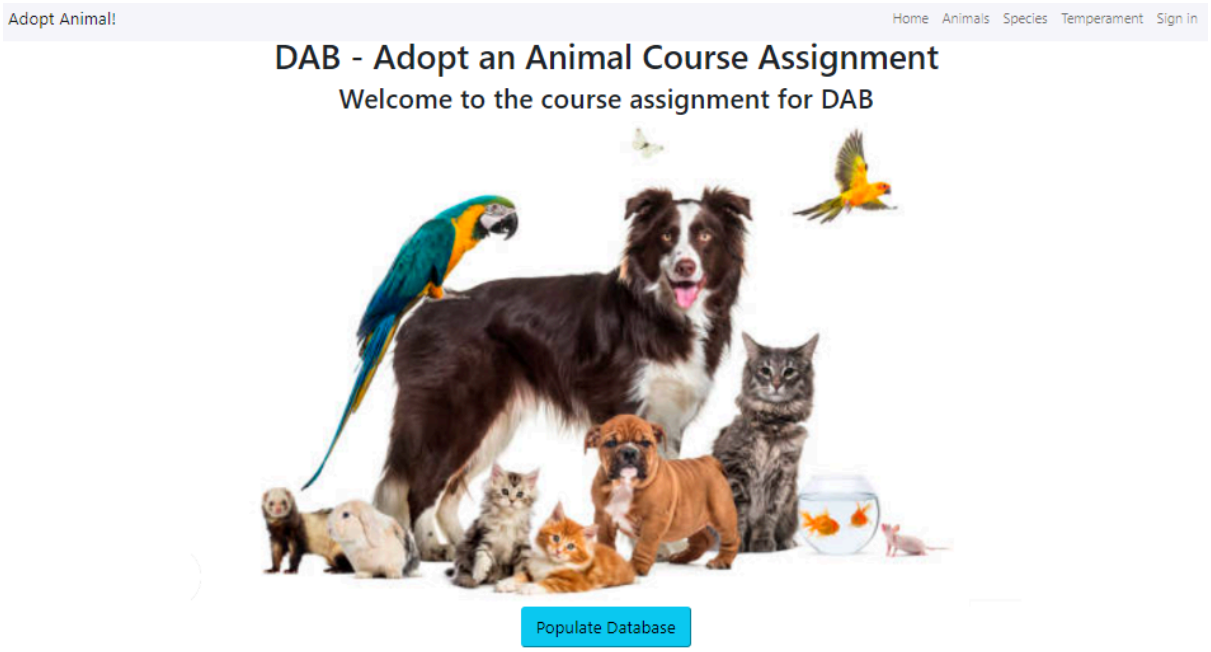
-TABLE CREATION-

Using Sequelize only:

Create all the necessary tables in the 3rd normal form (Including all relationships) from within the application: (Remember – the application should keep track of adoptions)

- One animal can have many temperaments. Many animals can have the same temperaments.
- An animal can only be one species and one size.
- A user can only have one role – either ‘admin’ or ‘member’.
- Each animal can only be adopted once.
- A single logged-in user of the ‘member’ role can adopt many animals.

-DATABASE POPULATION-



Clicking this "Populate Database" button on the Home page should populate the adoptiondb with the necessary data (if it has not already been populated).

Read data from JSON:

In the /public/json folder, create JSON files containing the population queries for each database table (One .json file per table) - This data can be found below in the tables.
Create Service files that use the JSON files to populate the database with the sequelize.query() function.
Make sure the database conforms to 3rd normal form.

HINT:

E.g., the example.json file would contain the query required to populate the Example table:

```
{
   id:1,
   query: "INSERT INTO TABLE VALUES..."
 }
```

As mentioned above, using Sequelize Raw SQL:

When the Populate Database button has been clicked, populate the necessary tables in the database by reading the SQL queries from JSON files.

Animals table						
Id	Name	Species	Birthday	Temperament	Size	Adopted
1	Coco	Dwarf Hamster	2020-02-12	calm, scared	small	FALSE
2	Ted	Teddy bear hamster	2021-02-12	calm, scared	small	FALSE
3	Coco	Jack-Russel	2020-02-12	energetic	medium	FALSE
4	Everest	Budgy	2019-02-12	calm, happy	small	FALSE
5	Rocko	Tortoise	2020-02-12	calm, lazy	medium	FALSE
6	Goldy	Gold Fish	2023-02-12	calm	small	FALSE
7	Lizzy	Lizzard	2020-02-12	calm,lazy	medium	FALSE
8	Goga	Bearded Dragon	2018-02-12	calm, lazy, scared	large	TRUE

9	Tweet Tweet	Parrot	2020-02-12	calm, happy	large	FALSE
10	Toothless	Corn snake	2017-02-12	scared	medium	FALSE
11	Sophie	Dwarf Hamster	2020-02-12	calm, scared	small	FALSE
12	Teddy	Teddy bear hamster	2021-02-12	calm, scared	small	FALSE
13	Roger	Parrot	2020-02-18	calm, happy	large	FALSE

Insert the following users into the correct table by reading the data from JSON files.

Users Table				
id	fullName	username	password	role
1	System admin	Admin	admin1234	admin
2	User	User	user1234	member
3	User2	User2	User1234	member

-DATABASE CREATION-

Save the following data creation SQL script to your README file, under the heading “DATABASE”.

Create the ‘adoptiondb’ database without any tables or data - Tables will need to be created from within the application using Sequelize (see instructions below).

-DATABASE ACCESS-

Save the following user-access SQL script to your README file, under the heading “DATABASEACCESS”.

Using SQL only:

- Create a new “dabcaowner” login for the database, with the password “dabca1234” with the “database owner” rights and permissions

-AUTHENTICATION-

The ‘users’ table should have the username and password for users that have signed-up for this service.

Make sure to:

- Implement the sign-up feature in the application/back-end
- PassportJS must be implemented to authenticate logged-in users, from the ‘users’ table. (Hashing of passwords is not required)

-ANIMAL ADOPTION-

All these routes must be created in the animals.js route file.
POST handlers should be used for data creation, updates, and additions, while GET handlers should be used to display a view.
Creation, deletion, and update operations must be authenticated.

Non-logged-in users (guest users) should not be able to adopt any animal.

- The “Animals” page should show all the animal records in the table.
- There must be an indication in the table of which animals have been adopted (Such as the ‘Adopted’ column).
- Only animals that have not already been adopted can be adopted.
- Remember, only logged-in users of the ‘member’ role can adopt animals.
- Only the logged-in user with the ‘admin’ role can cancel any adoption.
- When an adoption has been canceled, the relevant adoption record should be deleted from the database – do NOT delete the animal or the user.
- An animal’s age should be calculated using the Birthday field – do NOT store this age field in the database.

-SPECIES-

All these routes must be created in the species.js route file. All the endpoints need to be POST handlers, NOT GET. CRUD operations must be authenticated. Only users of the ‘admin’ role should be able to modify/add species data.

- A user with the ‘admin’ role can add a new species or update existing species’ names.
- A species cannot be deleted if there are any dependencies on that record in the database (e.g.: “Parrot” species cannot be deleted if there is an animal of species, “Parrot”).

-TEMPERAMENT-

All these routes must be created in the temperament.js route file. All the endpoints need to be POST handlers, NOT GET. CRUD operations must be authenticated. Only users of the ‘admin’ role should be able to modify/add temperament data.

- A user of the ‘admin’ role can add a new temperament or update existing temperaments.
- A temperament cannot be deleted if there are any dependencies on that record in the database (e.g.: “Calm” species cannot be deleted if there is an animal with the temperament “Calm”).

-DATABASE QUERIES-

Using Sequelize Raw SQL, create the following individual queries in the back-end for the respective buttons at the top of the “Animals” page.

- The “Animals for Adoption” table should be populated with the results of the queries after the buttons are clicked.
- Clicking the “All Animals” button will populate the table with all the animals in the Database.
- Ensure that your middleware functions contain authorization:
    - All users and guests can select the 'Popular Animal Names’ button.
    - All users and guests can select the 'All Adoption Details’ button.
    - All users and guests can select the 'Animals By Age’ button
    - All users and guests can select the 'Animals Born in Date Range’ button
    - Only the Admin user can select the 'Number of Animals per Size' button.

Further clarification on each SQL Query button's functionality:

Popular animals - Populate table with the most popular animal names (sorted from high to low)
All adoption details - Populate table with details of animals that have been Adopted
Animals by age - Populate table with all animals sorted by age
Animals born in date range - Populate table with filtered animals (E.g., a popup to receive user input is sufficient)

Number of Animals Per Size - Populating the table would require frontend editing, so an alert with the results would be sufficient.
(See the image above for an example of the table that is populated with the results of clicking the “All Animals” button).