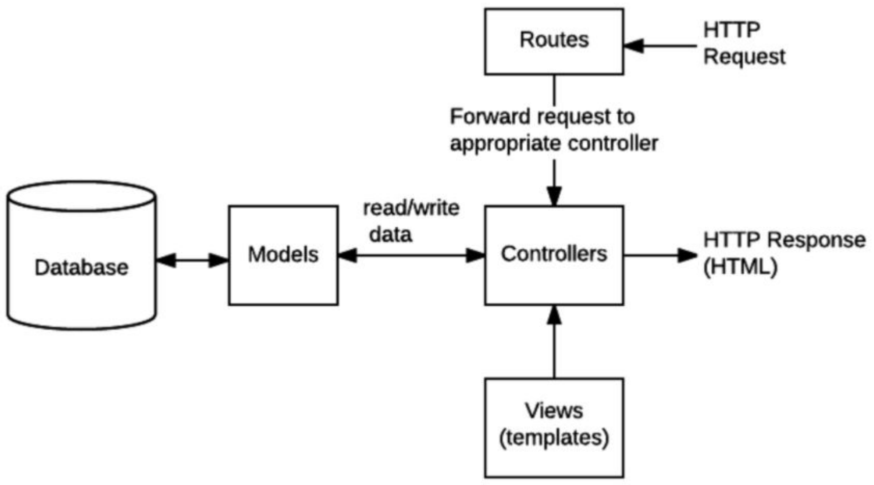
This assignment is to be completed individually. No group work is allowed.

This assignment is intended to familiarize students with serving views and defining routes web development. The goal is to use the MVC design pattern to structure your web application as MVC pattern is considered as the best practice. The assignment assumes student familiarity with static HTML design and will involve use of EJS template pages for the presentation, routes for the controller and JavaScript data objects for the business logic.

Assignment Description:

In this assignment, you will develop EJS/routes functionality using the MVC pattern, according to the following specifications:

1. Correct any errors identified in the selected HTML5 prototype before proceeding. - If your application developed in assignment 1 has error and not complete it is
   your responsibility to communicate and coordinate with course staff to help fix those bugs. Keep in mind that all that you are building in this application in stages. It is important that you have a working application for each stage.
2. All structure, design, and content requirements from the previous assignment are mandatory, unless explicitly updated in this assignment description.
3. Use JavaScript data objects to implement the business layer of the application (model).
4. Use EJS template pages to present the view to the browser.
5. Use routes to control the flow of the application.



<u>1. Convert Existing Pages to EJS Templates (View Prototypes)</u>

As an initial step, convert all 4 pages from the first assignment from HTML5 (.html) to EJS (.ejs).

- This is as simple as make a copy of the file with the ".ejs" extension.

Remember that HTML code for all pages generated from EJS must still meet the HTML5 validation requirement.

- You can use "view page source code" in your browser to see the rendered HTML response sent by the server for each view.
    - To validate view pages launch your application > navigate to the page you wish to validate > right click > view page source > copy all > paste into validator.w3.org 'Validate by Direct Input'.

<u>2. Create Module Views</u>

We want to centralize the common aspects of the views by creating partials for those common pieces and then adding (include) them where we need them.

- Separate the common design elements for header, footer, and navigation into individual .ejs files.

    - For example: header.ejs, navigation.ejs, footer.ejs
- For each of the 4 main pages
    - Remove the common design element content from the page
    - Use the EJS include to re-include that content from the individual files.

3. Add New Template Site Pages Linked from General Navigation

If not already part of the project, add the following pages using the required site structure.

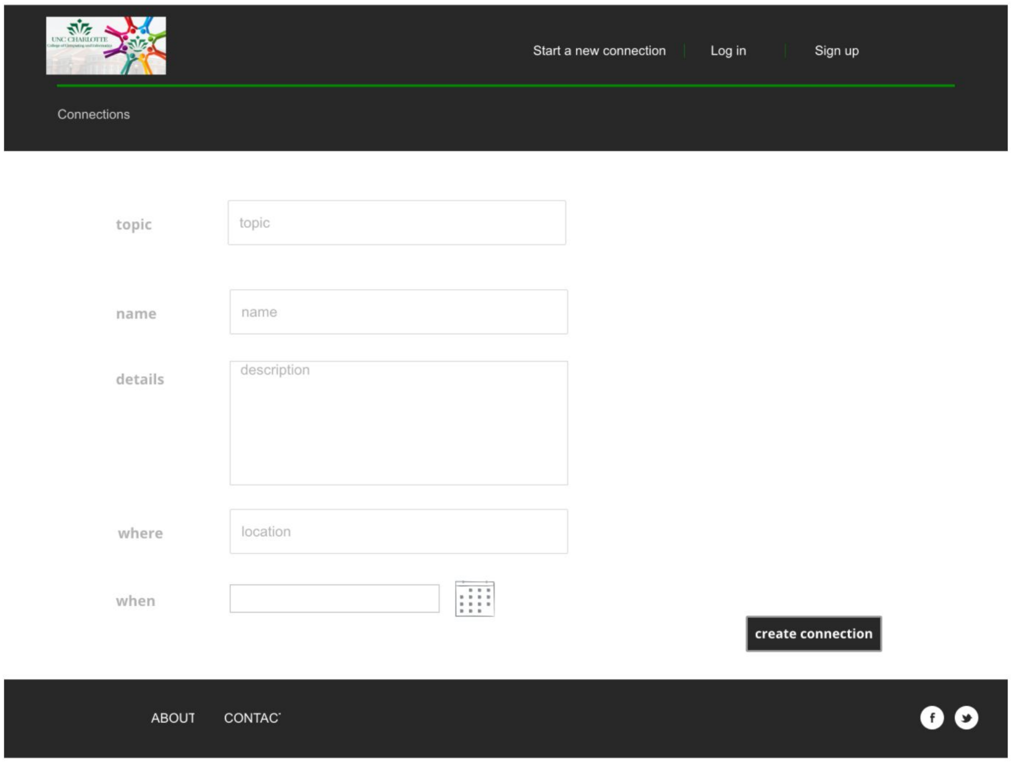
- about.ejs – provides background and detail about the application site.
- contact.ejs – provides contact details for the application site.

Update links to About and Contact to render those views.

4. Add New Template Site Page for Creating a Connection

Create a new (static, design prototype) EJS template page that allows a user to create a new connection.

Filename: newConnection.ejs



5. Create Data Objects (Model)

This assignment will make use of simple JavaScript objects for the Model, in order to represent the main data elements / business objects being used.

Create JavaScript objects (connection.js) to represent a connection as a Model element with specific properties.

This model is used to represent a connection in the application. It should include the properties/attributes that define a connection as well as methods/functions to get and set those properties (getters and setters)

- connection ID– unique identifier for the connection, you should select a sensible format (e.g., alphanumeric string with some specific pattern) that will be standard across your connections.
- connection name
- connection topic - this is the category/topic that will be used to select or arrange connection by section or type in the connections view.
- details
- date and time

6. Create Utility Functions for Persistent Data

Create a utility module to support data retrieval called connectionDB with following requirements and functions:

- Includes a hard-coded set of connection details (your choice on how to represent internally, but must be converted into a list/collection of connection objects when required)
- Includes a function called getConnections() that returns a set of all the connections in the hardcoded "database"
- Includes a function called getConnection(connection ID) that returns the connection with the specified ID from the hardcoded "database" . Rather than having the controller filter the complete connection list each time a connection is requested you should encapsulate this functionality in a function that can me called when needed.

For initial development, we will use a hard-coded "database" to represent your initial set of connections. You should hard-code a fixed set of connections within the following module. Do this through creating functions that return connection objects.

As in the first assignment, you must have at least 2 categories of connections, with at least 3 connections per category/topic. Note that this means you will need to create the individual data for the additional connections, since there was only individual data for one representative connection required in the first assignment.

Note: It is important to keep the big picture in perspective here. In part 4 of this application we will be converting the data storage to a database platform. We will refactor these functions to support database connectivity and query. This will separate the data layer from the application and will allow for easy updates and interaction. At this stage we just need to read/retrieve the data so we will read it from a function.

7. Create Routes for Business Logic (Controller)

For this assignment, you will implement a limited part of the dynamic functionality for the connections and connection views. You must use ONLY the http request object to pass data.

Implement the following controller logic to operationalize the business logic. You will need to send parameters as part of the GET or POST http requests from link / button / form submissions as the context information that tells the controller how to proceed.

Connection Controller

This controller is responsible for handling and directing requests that pertain to displaying the categories of connections (catalog) and individual connection details. An application user should be able to navigate to a view that lists all connections saved in the application (connections.ejs) as well as navigate to a page providing connection details from the connections page (connection.ejs). The connection controller will be utilized to create the flow of the application.

This route controller is responsible for handling the following two requests:

- for a request to view the complete list of connections this controller responds by bringing all connection models from the database and sending that data to the connections view for display.
- for a request to view a single connection this controller responds by bringing the requested connection model from the database and sending that data to the connection view for display.

Part of handling these request is addressing scenarios when a request comes in that doesn't meet the requirements for these requests. The controller logic should verify and validate each request. If there are any parameters coming in with the request the controller should verify that they contain values that are expected by the application and it can handle. For example if a request comes in with a connection code and that value is not for a connection code that exists in the connection database the controller should gracefully handle it in a way that will allow the user to continue interacting with the application. In this example, displaying the connections view would be a valid response.

As we are starting to put logic into the application and connecting this logic to the different views (webpages) we've designed it is important to know and keep in mind that users can interact (send requests and receive responses) from any application that in on the world wide web without necessarily using the webpages that are tied to the application. A user can simply send a request to a web-based application using command line for example. Do not rely on users using only the links that are implemented in the application views.

Sample Basic Pseudocode:

- Loads the catalog / database of connections.
    - save the catalog of connections by calling the getConnections() function
- Checks the http request for a parameter called “connectionID”
    - If there is a connectionID parameter, validate that its value matches your connection ID format and is a valid connection ID.
    - If the connection ID is valid
        - Get the object for the specified connection to have it available for the views to render
        - Dispatch to the correct EJS view for **individual connection display**
    - If the connection code is not valid, display the catalog as if no code had been provided
    - If there is no connection ID parameter, dispatch to the correct EJS view for **connections display**. Display the full connections listing, separated by category/topic as in the first assignment, but with entries created dynamically from the connections list

8. Update EJS Template Views to Include Dynamic Content From Data Object

The connections and connection views will now receive some kind of dynamic data. Update the EJS template views to replace all of the static placeholder information with the dynamic data using EJS template functionality to access object data being passed from the controller. The connections views should handle and display the connections categories and the connections titles/links dynamically. In other words, if you add another connection to the hard coded list with a new category the application will be able to handle that without having to update or change any code.

9. Update Form / Button Actions and Links in the Site to Dispatch Correctly

Each of the places in the site where a user can take an action that uses the dynamic data from this assignment should be updated to make the appropriate GET or POST request with the necessary parameters or form data. Remember that requests for resources such as links should be using GET method and any requests that sends data such as form submissions must use POST. Updates must be made in all appropriate places where action would reasonably be indicated from the first assignment prototypes. For example: the links to a connection in connections.ejs will need to be dynamic to load the correct connections.

Assignment Submissions

What to submit using Canvas (Email submissions will NOT be accepted):

1. **Milestone2.zip** - An archive of the entire web application (project) stored in a standard ZIP File, you must ensure that all the files are included as part of the archive. **Do NOT** use another compression format such as .rar or .7zip or you could get a 0 on your assignment.
2. **Milestone2Info.pdf** – PDF document with the following assignment information:
    1. Screenshots of the all views of your application displayed in your browser.
    2. Explanation of additional features, if any.
    3. Explanation of status, stopping point, and issues if incomplete.
    4. Discuss the easy and challenging parts of the assignment. How did you overcome all or some of the challenges?