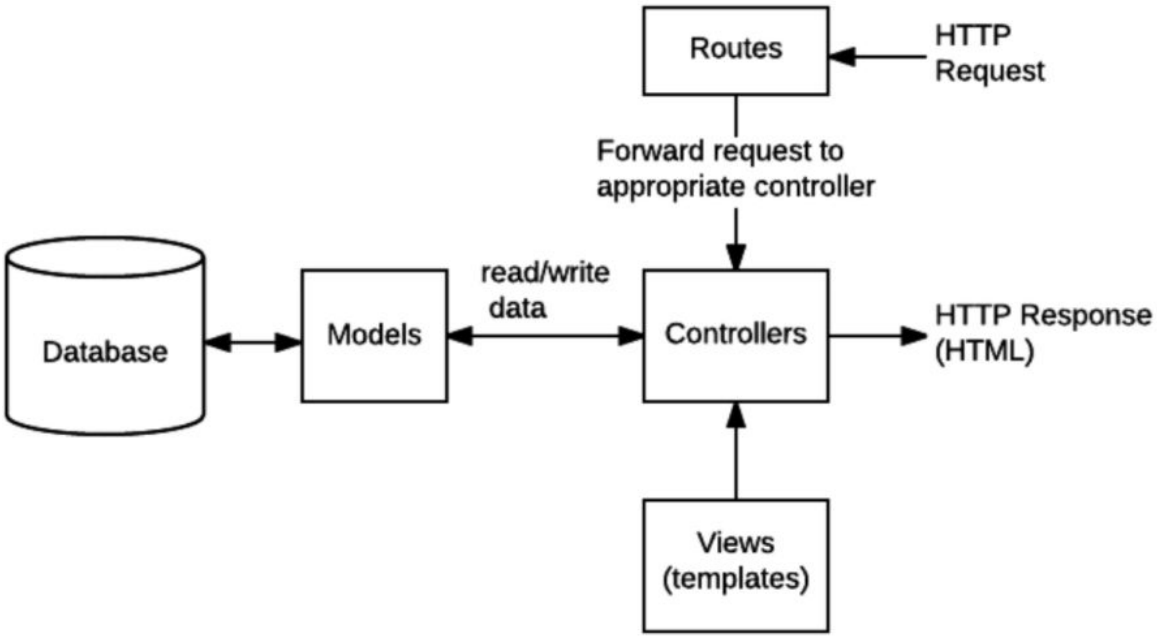
Put Milestone 1, 2, 3, 4 together

- Use JavaScript to implement the business layer of the application (**model**).
- Use EJS pages to present the **view** to the browser.
- Use routes to **control** the flow of the application.
- Functionality that does not follow the milestone specifications will not receive credit.



The goal is to implement logic to support the following functionalities:

- user responding (RSVP) to a connection to save to their account (profile)
- user removing (delete) connections they've saved in their account (profile)
- user changing (update) their response. A user should be able to change any response they previously provided.
-

1. Create New Objects for Business Objects (Model)

Create Javascript objects for the following Model elements, with the specified properties and additional functions/methods.

User - represents a user of the application with the following properties:

- User ID
- First Name
- Last Name
- Email Address
- Optional fields
    - Address 1 Field
    - Address 2 Field
    - City
    - State
    - Zip Code
    - Country
- Any other fields you find necessary

UserConnection – represents a connection object saved to the user profile (associates a connection to a user profile) with the following properties:

- Connection
- rsvp
- Any other fields you find necessary (optional)

2. Create a utility class/module to manage a user's profile

UserProfile - represents the user profile. Contains the list of user connections as well as functions to support getting, adding and removing connections from the user profile. In other words, it's a class/module with methods/functions to store and manage the user connections in the user profile.

- properties:
    - User ID or User to associate this UserProfile object to the user
    - a list containing UserConnection objects for this user
- methods/functions
    - addConnection – adds a UserConnection for this connection / rsvp to the user profile. The profile should not allow multiple UserConnections for the same connection, but should update appropriately if one already exists.
    - removeConnection– removes the UserConnection associated with the given connection.
    - updateRSVP- updates an RSVP property for a specified UserConnection
    - getUserConnections – returns a List / Collection of UserConnection from the user profile

- Any other methods/functions you find necessary

3. Create Routes for Business Logic (Controller)

For this milestone, you will implement dynamic user profile functionality. You must use sessions to store data. This means that the user profile will be able to hold multiple connections at a time, and will maintain its content even if you navigate away from the user profile page (savedConnections view). We will be referring to the page with a user's saved connections as the **profile** view (think of this view as the user's dashboard. It displays a summary view of the connections they saved/shared and provides them ways to manage their profile).

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 button / form submissions as the context information that tells the controllers how to proceed.

UserController

(This can be split into multiple controllers, **but you must specify in your Milestone3_Info.pdf how you have divided the tasks or risk losing points**).

This controller is responsible for answering requests that pertain to user-specific functionality. These are the requests for accessing a service the application provides to support its users. These are registered users (we are simulating that with are hardcoded data) and these services/functionalities are specific and available to them. An application-user should be able to navigate to a view that lists connections they saved in the application as well as navigate to a view to provide connection RSVP.

Functionality:

- Login a user (add a user to the session without email/password verification) by initializing a UserProfile model and storing it in the session.
- This controller should continue to store a user's current profile contents in the session as long as the session is not destroyed
- Save a user's rsvp for a connection - add a new connection to the user profile
- Update a user's rsvp for a connection - update the value for the RSVP for a specific user connection already in the profile
- Delate a user's rsvp for a connection - delete a specific user connection already in the profile
- Display a user's list of saved connections - list all user connections on the savedConnections view
- Logout a user (remove a user from the session)

4. Database Creation

In addition to the standard project folder for this milestone, you will submit a text file (plain text, NOT Word or RTF or PDF or any other fancy document format). This file will contain all of the MongoDB queries that you use to (1) create, and (2) populate your database. Your database script file must be called:

milestone4_createDB.txt

Create a database in MongoDB to hold your application data. Create the necessary documents and fields to store the information needed in the application (e.g., users, connection, userConnection).

5. Database Population

Your database should be populated with data according to the same requirements from the previous milestones (at least two topics of connections, with at least 3 connection per topic/category).

Within the same script file for database and table creation (milestone4_createDB.txt), add statements at the end of the script to insert your initial data into each of the two tables for connections and users (see #6 for adding a user).

Running the statements in milestone4_createDB.txt script should re-create / reset your database to its initial state with initial data (this means that at any point we can revert back to the beginning as a starting point).

ConnectionDB - Refactor the “ConnectionDB” module to retrieve connections from the database:

- getConnections() - this function returns an array of Connection objects of all the connections in the connections table from the database
- getConnection(connectionID) - this function returns a Connection object for the provided connection code

UserDB - create the module “UserDB” to retrieve a user from the database:

- getUser(email/userID) - this function returns a User object for the provided user identifier

UserProfileDB – create the module “UserProfileDB” to save and retrieve connections to users and save any feedback they give for connections in the database. This module should provide/support the following functionality:

- retrieve all connections a user has RSVP'd to (a list of UserConnection objects)
- add/update a user RSVP to a connection
- add a new connection to the list of available connections

Sample design:

- getUserProfile(email/userID) - this function returns a collection/list of all UserConnection objects saved
- addRSVP(connectionID, userID, rsvp) – this function associates a connection to the user with userID (when the user saves the connection) and updates the RSVP that user provides for a connection with this connection ID. The corresponding database document can use the userID and connectionID as unique identifiers (keys).
    - **Note**: you need to be able to distinguish between connections that belong to the user (they shared it) vs. connections they RSVP to. If you have not done so already, add a user ID field to each connection in the database. This will allow you to tell which users own which connections.
- updateRSVP(connectionID, userID, rsvp) – this function updates the rsvp field for a user with userID for connection with connectionID.
- addConnection(connection) - this function handles the new connection form and adds a new connection to the list of connections in the database.

6: Add Logic to Handle User Login

- When the user clicks the sign in button this should send a request to verify that username and password submitted belong to an already registered user.
- If the user credentials do not validate, the login view should be displayed showing an error message. (e.g. Either username or password are incorrect. Please try again.)
- If the user credentials are verified display the profile view.
- **Note** that this might require updating your models and database to account for username and password. Your choice on how and where to handle representing these properties in the models and the database (e.g., adding to existing models and documents or creating new models/documents specifically for login info)

Pseudocode:

- If the action parameter validates to a value of “signIn”:
    - Check the session for “theUser” attribute
    - If a “theUser” attribute is set dispatch to **profile** view (savedConnections.ejs).
    - If there is no valid User object stored in the session
        - Checks the http request for a parameter called “username”

    - If there is no username parameter, or it has an unknown value dispatch to the login view.
    - If there is a username parameter and is valid check the http request for a parameter called “password”
    - If there is no password parameter, or it has an unknown value dispatch to the login view.
    - If there is a valid username and password, verify the username and password matches a valid user in the database.
    - If the user credentials are valid retrieve the list of saved connections for that user from the database. If a user doesn't have any saved connections this should be empty.
    - Add the list of saved connections to the session object as “currentProfile”.
    - Dispatch to the **profile** view
    -

7: Applying Security Measures

- Update your application to make use of the express-validator package to validate all input values read into the application (login and add connection)
- Update your application to escape dynamic output displayed in all views