Introduction

This assignment is the fourth of six assignments. It has been designed to give you practical experience working with no-sql databases.

Before you begin this assignment, you must finish your previous assignment. All objectives listed for this assignment are to be made "on top" of your previous assignment.

This assignment is worth 9% of your final grade.

Reminder about academic integrity

You must comply with Seneca College's Academic Integrity Policy. Although you may interact and collaborate with others, this assignment must be worked on individually and you must submit your own work.

You are responsible to ensure that your solution, or any part of it, is not duplicated by another student. If you choose to push your source code to a source control repository, such as GIT, ensure that you have made that repository private.

A suspected violation will be filed with the Academic Integrity Committee and may result in a grade of zero on this assignment or a failing grade in this course.

Technical Requirements

- All back-end functionality **must** be done using **Node.js** and **Express**.
- You will use the **body-parser** module to handle form submissions.
- You will use the **express-session** module to handle user session state information.
- You will use **bcrypt.js** to encrypt user passwords.
- Your views **must** be created with **Express-Handlebars**.
- You **can use** a front-end CSS framework such as Bootstrap, Bulma or Materialize CSS to make your website responsive and aesthetically pleasing.
- You are **not allowed** to use any Front-End JavaScript Frameworks. For example, you may not use React, Vue, or Angular.
- You **must** use MongoDB as your database engine.

Objectives

MVC Design Pattern

Your application **must** be structured according to the MVC Design Pattern. In other words, your views, models, and controllers must be separated as per the design pattern requirements learned during lectures.

In a previous assignment, you created a "fake" database to store meal kit information. You can leave your meal kits database "as is" for now. You will move meal kits into a Mongo DB during a future assignment.

Sensitive Information

In the previous assignment, you stored sensitive information, such as the SendGrid API key, in environment variables. You will continue this trend by securing the MongoDB connection string. Do not forget, this file should not appear on GitHub but must be submitted on Blackboard for testing.

User Registration Module

You are required to add database functionality to the registration page that was implemented in the previous assignments. When a user fills out the registration form and presses the submit button:

- check that all validation criteria have passed validation *(already implemented in previous assignment)*.
- create a user account in your database.
- redirect the user to a welcome page *(already implemented in previous assignment)*.

With respect to database functionality, the following rules must be followed:

1. Setup and configure a MongoDB cloud service using MongoDB Atlas.
2. Connect your web application to your MongoDB database using an Object Document Mapper (ODM) called Mongoose.
3. Name your database and collections appropriately.
4. Ensure that the email field in your registration form is unique. Your application must prohibit different users from having the same email in the database.
5. Passwords must not be stored in plain text in the database. Your application must store passwords in an encrypted format. You can use a 3rd party package called bcrypt.js to help you.

Authentication Module

You are required to implement a fully functional authentication module with the following features:

- Your application must allow two types of logins. Add a checkbox (or radio buttons) to allow the user to specify the role they will sign in as. A user may choose to login as a "data entry clerk" or a "customer".
    - Data Entry Clerk – users that can add, remove, and modify meal kits.
    - Customer – users that can purchase meal kits.
- Upon successful authentication (entering an email and password pair that exists in the database) a session is created to maintain user state until the user has logged out of the application. For help on implementing sessions in an Express app, read the following article: https://github.com/expressjs/session.
- Upon an unsuccessful authentication, the application must display an appropriate message. For example, "Sorry, you entered an invalid email and/or password".
- After successfully authenticating, the application must determine if the person logging in is a data entry clerk or a regular user and will redirect them to their respective dashboard.
- A customer will be directed to a customer dashboard and a data entry clerk will be directed to a data entry clerk dashboard. Each dashboard is considered a different page/route.
- Both dashboards, must show the user's name (first name and last name) and a logout link. Alternatively, this information can be shown in the navigation bar.
- The logout link must destroy the session created when the user initially authenticated.
- Specific routes can only be accessed when users are logged-in, thus those routes must be protected. A customer cannot access the data entry dashboard and the data clerk cannot access the customer dashboard. A data entry clerk will not purchase meal kits.

GitHub

You will continue to push your web application to a remote GitHub repository in your own account. **Do not forget to set your remote repository to private.** Add your professor as a collaborator so he/she can view your web application.

A realistic view of your progress must be showed by looking at your commits.

Rubric

Criteria	**Not Implemented (0)** Little or no work done. Unacceptable attempt.	**Partially Implemented (1)** Work is minimally acceptable but is incomplete or needs significant modification.	**Fully Implemented (2)** Work is complete and done perfectly.
User Registration • MongoDB cloud service is setup. • Database and collection have appropriate names. • User data is inserted into the database when the user fills out the form and hits the submit button. • Email uniqueness validation. • Password is stored in encrypted format.			

Authentication (Sign In) • After an unsuccessful authentication, the application displays an appropriate error message. • After success authentication, a session is created to maintain user state. The session exists until the user logs out of the application. • The application directs a data clerk to their dashboard and a regular user to his or her dashboard. • Both dashboards, show the user's name (first name and last name) and a logout link. • Dashboards can only be accessed when users are logged on and by users with the appropriate user roles. • The logout link destroys the session.			
Architecture • The source code has been structured using the MVC techniques learned in class.			

Total: 24 Marks

Note: Half marks may be awarded.

Submitting your work

Make sure you submit your assignment before the due date and time. It will take a few minutes to package up your project so make sure you give yourself a bit of time to submit the assignment.

1. Locate the folder that holds your solution files.

2. Compress the copied folder into a zip file. **You must use ZIP compression, do not use 7z or other compression algorithms or your assignment will not be marked.**

3. Login to My.Seneca.

4. Open the **Web Programming Tools and Frameworks** course area and click the **Project** link on the left-side navigator. Follow the link for this assignment.

5. Submit/upload your zip file. The page will accept three submissions so you may re-upload the project if you need to make changes. Make sure you make all your changes before the due date. Only the latest submission will be marked.