Regex, Express, and Full-Stack Development (Week 11) - Learning Objectives

Assessment Structure

- 2 hours, 45 minutes
- Mixture of multiple choice (5-10), and VSCode problems (20-30 specs).
  - For the coding section, be comfortable with creating an express application from scratch. The projects from Thursday and Friday are good examples (on a smaller scale for an assessment environment).
  - This includes connecting database material introduced last week, such as making your database, initializing sequelize, creating migrations and models, adding seeders for your models, etc.
  - Be able to work within the express framework, creating the app, adding in middleware for csrf protection, urlencoded body parsing, etc., making route handlers to respond to various requests and pull in database information, etc.
  - Be able to display that information to the user using Pug templates, including csrf-protected forms for post routes, iterating over database data to display content of records, etc.
- Standard assessment procedures
  - You will be in an individual breakout room
  - Use a single monitor and share your screen
  - Only have open those resources needed to complete the assessment:
    - Zoom
    - VSCode
    - Browser with AAO and Progress Tracker (to ask questions)
    - Approved Resources for this assessment:
      - Express Docs: http://expressjs.com/
      - Pug Docs: http://pugjs.org/
      - csurf Docs: https://github.com/expressjs/csurf#readme
      - Sequelize Docs: https://sequelize.org/
      - Sequelize "Cheatsheet"

Regular Expressions and Node HTTP (W11D1) - Learning Objectives

Regular Expressions

1. Define the effect of the * operator and use it in a regular expression
2. Define the effect of the ? operator and use it in a regular expression
3. Define the effect of the + operator and use it in a regular expression
4. Define the effect of the . operator and use it in a regular expression
5. Define the effect of the ^ operator and use it in a regular expression
6. Define the effect of the $ operator and use it in a regular expression
7. Define the effect of the [] bracket expression and use it in a regular expression
8. Define the effect of the - inside brackets and use it in a regular expression
9. Define the effect of the ^ inside brackets and use it in a regular expression

Node HTTP (HTTP Full-Stack)

1. Identify the five parts of a URL
2. Identify at least three protocols handled by the browser
3. Use an IncomingMessage object to

- access the headers sent by a client (like a Web browser) as part of the HTTP request
- access the HTTP method of the request
- access the path of the request
- access and read the stream of content for requests that have a body

4. Use a ServerResponse object to

- write the status code, message, and headers for an HTTP response
- write the content of the body of the response
- properly end the response to indicate to the client (like a Web browser) that all content has been written

Express and Pug Templates (W11D2) - Learning Objectives

Express

1. Send plain text responses for any HTTP request
2. Use pattern matching to match HTTP request paths to route handlers
3. Use the Pug template engine to generate HTML from Pug templates to send to the browser
4. Pass data to Pug templates to generate dynamic content
5. Use the `Router` class to modularize the definition of routes

Pug Templates

1. Declare HTML tags and their associated ids, classes, attributes, and content
2. Use conditional statements to determine whether or not to render a block
3. Use interpolation to mix static text and dynamic values in content and attributes
4. Use iteration to generate multiple blocks of HTML based on data provided to the template

HTML Forms (W11D3) - Learning Objectives

HTML Forms

1. Describe the interaction between the client and server when an HTML form is loaded into the browser, the user submits it, and the server processes it
2. Create an HTML form using the Pug template engine
3. Use express to handle a form's POST request
4. Use the built-in `express.urlencoded()` middleware function to parse incoming request body form data
5. Explain what data validation is and why it's necessary for the server to validate incoming data
6. Validate user-provided data from within an Express route handler function
7. Write a custom middleware function that validates user-provided data
8. Use the csurf middleware to embed a token value in forms to protect against Cross-Site Request Forgery exploits

Full-Stack (Data-Driven Web Sites) (W11D4) - Learning Objectives

Data-Driven Web Sites

1. Use environment variables to specify configuration of or provide sensitive information for your code
2. Use the `dotenv` npm package to load environment variables defined in an `.env` file
3. Recall that Express cannot process unhandled Promise rejections from within route handler (or middleware) functions
4. Use a Promise `catch` block or a `try`/`catch` statement with `async`/`await` to properly handle errors thrown from within an asynchronous route handler (or middleware) function
5. Write a wrapper function to simplify catching errors thrown within asynchronous route handler (or middleware) functions
6. Use the `morgan` npm package to log requests to the terminal window to assist with auditing and debugging
7. Add support for the Bootstrap front-end component library to a Pug layout template
8. Install and configure Sequelize within an Express application.
9. Use Sequelize to test the connection to a database before starting the HTTP server on application startup
10. Define a collection of routes (and views) that perform CRUD operations against a single resource using Sequelize
11. Handle Sequelize validation errors when users are attempting to create or update data and display error messages to the user so that they can resolve any data quality issues
12. Describe how an Express.js error handler function differs from middleware and route handler functions
13. Define a global Express.js error-handling function to catch and process unhandled errors
14. Define a middleware function to handle requests for unknown routes by returning a 404 NOT FOUND error