Web Programming

Tutorial 8

To begin this tutorial, please create a Node project or use an existing one. When you finish, zip all your source codes (excluding the `node_modules` folder) to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `tut01`, `tut02`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Activity 1 – Crypto Module

In this exercise, you'll work with the Node.js crypto module to implement encryption and decryption using the `AES-128-ECB` mode. This mode is a straightforward encryption technique where each block of plaintext is encrypted separately.

Requirements:

1. **Create Encryption and Decryption Functions**:
    - **Encryption Function**:
        - Create a function `encryptText(plainText, key)` that encrypts the provided `plainText` using the provided `key`.
        - Use the `AES-128-ECB` algorithm for encryption.
        - Return the encrypted text in `base64` format.
    - **Decryption Function**:
        - Create a function `decryptText(encryptedText, key)` that decrypts the provided `encryptedText` using the provided `key`.
        - Return the decrypted plain text.
2. **Key length**:
    - The key length for `AES-128-ECB` is 16 bytes. Ensure that the provided key is exactly 16 bytes long (128 bits).
3. **Test the above functions**

- Write simple Node.js program to test the `encryptText` and `decryptText` functions.

4. **Put these functions in a CommonJS module and re-use them in the next activities.**

Activity 2 – User Login with EJS and Cookies

In this exercise, you'll implement a user login feature using JSON to manage user data, EJS as template engine, `cookie-parser` and `crypto`. By completing this exercise, you will gain practical experience in building and securing web applications, handling user data, and managing user sessions. These skills are fundamental for developing robust and secure web applications.

Requirements:

1. **Create a JSON File**:
    - Create a JSON file, for example `users.json`, to store user information. The JSON file should include the following fields for each user:
        - `id (User ID)`
        - `name (User Name)`
        - `username (Username)`
        - `password (Password)`
        - `avatar (Path to profile picture)`
        - `signupTime (Signup Time)`
2. **Create Endpoints**:
    - **GET Endpoint** `/login`: Create an endpoint to display the login form. This form should allow users to enter their `username` and `password`.
    - **POST Endpoint** `/login`: Create an endpoint to handle the login form submission. This endpoint should check the user information against the `JSON` file and respond with error messages if the input is incorrect (e.g., *user not found*, *wrong password*). If the login is successful, create a `cookie` named user with the user's ID and redirect to `/profile`.

3. **Create Middleware**:
    - Create middleware to decrypt all cookies from the `req.cookies` object.
    - Override the `res.cookie` function to encrypt all cookies before sending them to the client.
4. **Create EJS Templates**:
    - `login.ejs`: Create an EJS template to display the login form and any error messages.
    - `profile.ejs:` Create an EJS template to display user profile information. The `/profile` endpoint will show the user's information based on the user cookie.

Activity 3 – User Login - Handlebars Template

In this activity, you will implement a user login feature similar to the previous activity but using Handlebars instead of EJS for template rendering. This task will help you apply the same core concepts with a different **templating engine**, which is valuable for understanding different tools in web development.

- **Template Syntax**: Replace EJS syntax `(<%= %>`, `<% %>`, etc.) with Handlebars syntax (`{{ }}`, `{{#if }}`, etc.).
- **Error Handling and Data Display**: Update the login form and error display logic to use Handlebars expressions and helpers.

Activity 4 – User Login Feature with MVC Model

In this exercise, you will refactor your existing user login feature by applying the **Model-View-Controller (MVC)** design pattern. This pattern helps in organizing your code into distinct sections, making it easier to manage and scale. You'll move your existing code into an MVC structure and understand how this architectural pattern improves code organization.

Requirements:

1. **Refactor the Code**:
    - **Model**: Create a model to handle user data interactions. This model will include functions to read from and validate against the `users.json` file.

    - **View**: Separate the Handlebars templates into a views directory. Ensure your views (`login.hbs` and `profile.hbs`) only contain presentation logic.
    - **Controller**: Develop controllers to handle user input, business logic, and interaction between models and views. This will include:
        - `UserController`: Handle login requests, validate credentials, set cookies, and manage redirects.
        - **Middleware**: Create middleware for cookie encryption/decryption.
2. **Update Routes**:
    - Refactor the existing route handlers to use the new controllers. Ensure routes handle requests by calling appropriate methods in the controllers.
3. **MVC Structure**:
    - **Model**: Contains the business logic and data handling.
        - **File**: `models/userModel.js`
    - **View**: Contains the presentation layer.
        - **Files**: `views/login.hbs`, `views/profile.hbs`
    - **Controller**: Contains the logic to manage user requests and interact with models and views.
        - **File**: `controllers/userController.js`
    - **Middleware**: Manage cookies and security aspects.
        - **File**: `middleware/cookieMiddleware.js`