**CSE 340 – Outcome Mastery Report**

**Name**: Favour Ojobor  
**GitHub URL**: https://github.com/Yeva04/cse340.git  
**Deployed website URL**: https://cse340-4wcp.onrender.com

**Instructions**:  
For each of the course outcomes listed below, include 2-3 examples of the way you have demonstrated this outcome. Each example should include:  
· A reference to the specific file/function you have created.  
· A 2-3 sentence description of what you did.

**Outcome 1: Develop to current web frontend standards of validity and practice.**

**Examples**:

1. **Example**: I implemented valid HTML and CSS for the vehicle detail view with sticky form inputs for the add inventory form.  
   a. **File/Function**: views/inventory/detail.ejs, public/css/styles.css  
   b. **Description**: Created detail.ejs with valid HTML5, including semantic elements like <h1>, partials, and styled it with styles.css using flexbox for responsive layouts (e.g., .vehicle-detail). Ensured accessibility with proper <label> elements in the add-inventory.ejs form, which was made sticky to retain input values on validation errors, meeting frontend checklist standards.
2. **Example**: I developed a management view with flash message display and responsive design.  
   a. **File/Function**: views/inventory/management.ejs, public/css/styles.css  
   b. **Description**: Built management.ejs to display flash messages for successful/failed operations (e.g., adding classifications) using valid HTML5 and styled it with CSS to ensure responsiveness across devices. The view includes links to add classifications and inventory, adhering to frontend best practices like clear navigation and consistent styling.
3. **Example**: Updated header partial to dynamically show/hide links based on login state.  
   a. **File/Function**: views/partials/header.ejs  
   b. **Description**: Modified header.ejs to conditionally render "My Account" or "Logout" links based on JWT authentication status, ensuring links are absent from the DOM when hidden. Styled the header with styles.css to maintain consistent design, meeting accessibility and validity standards.

**Outcome 2: Use variables, arrays, functions, and control structures in server code.**

**Examples**:

1. **Example**: I built a controller function to fetch and display vehicle details.  
   a. **File/Function**: controllers/invController.js/buildByInventoryId  
   b. **Description**: Used variables and control structures in buildByInventoryId to parse inv\_id, fetch vehicle data via getInventoryById, and conditionally set the page title based on the result. The function handles errors with a try-catch block, ensuring robust server-side logic.
2. **Example**: Created a utility function to build a dynamic classification select list.  
   a. **File/Function**: utilities/index.js/buildClassificationList  
   b. **Description**: I developed buildClassificationList to query classifications, store them in an array, and use a forEach loop to generate a <select> element with options. The function uses conditional logic to pre-select the current classification, enhancing form usability.
3. **Example**: Implemented account update logic with variable handling.  
   a. **File/Function**: controllers/accountController.js/updateAccount  
   b. **Description**: Wrote updateAccount to extract account\_id, account\_firstname, account\_lastname, and account\_email from req.body using destructuring. Used control structures to validate inputs and update the database, setting success/failure messages stored in session variables.

**Outcome 3: Develop web applications that implement common design patterns.**

**Examples**:

1. **Example**: Applied MVC architecture for vehicle detail view rendering.  
   a. **File/Function**: routes/inventoryRoute.js, controllers/invController.js/buildByInventoryId, models/inventory-model.js/getInventoryById, views/inventory/detail.ejs  
   b. **Description**: Implemented MVC by routing requests through inventoryRoute.js, processing logic in buildByInventoryId to fetch vehicle data via getInventoryById, and rendering detail.ejs. This separated concerns, with the model handling database queries, controller managing logic, and view displaying data.
2. **Example**: Used MVC for adding new classifications with error handling.  
   a. **File/Function**: routes/inventoryRoute.js, controllers/invController.js/addClassification, models/inventory-model.js/insertClassification, views/inventory/add-classification.ejs  
   b. **Description**: Designed an MVC process where inventoryRoute.js routes POST requests to addClassification, which validates input and calls insertClassification to add to the database. The view add-classification.ejs displays validation errors or success messages, following the MVC pattern.
3. **Example**: Implemented account management with JWT middleware.  
   a. **File/Function**: routes/accountRoute.js, controllers/accountController.js, middleware/jwtAuth.js, views/account/management.ejs  
   b. **Description**: Built an MVC process for account management, using jwtAuth.js middleware to restrict access to Employee/Admin users. Routes in accountRoute.js call accountController.js functions to render management.ejs with dynamic greetings and links, adhering to MVC principles.

**Outcome 4: Design and use relational databases for CRUD interactions.**

**Examples**:

1. **Example**: Retrieved vehicle data for classification and detail views.  
   a. **File/Function**: models/inventory-model.js/getInventoryByClassificationId, getInventoryById  
   b. **Description**: Created getInventoryByClassificationId to query the inventory and classification tables using a JOIN to fetch vehicles by classification\_id, returning data.rows. Similarly, getInventoryById retrieves a single vehicle by inv\_id using a parameterized query, ensuring safe CRUD read operations.
2. **Example**: Inserted new classifications into the database.  
   a. **File/Function**: models/inventory-model.js/insertClassification  
   b. **Description**: Developed insertClassification to insert a new classification name into the classification table using a parameterized query to prevent SQL injection. The function returns the insertion result to the controller for success/failure messaging.
3. **Example**: Updated account information based on account\_id.  
   a. **File/Function**: models/account-model.js/updateAccount  
   b. **Description**: Implemented updateAccount to update account\_firstname, account\_lastname, and account\_email in the account table using a parameterized query with account\_id. The function returns the update result, supporting secure CRUD update operations.

**Outcome 5: Validate data (client-side and server-side) appropriate to the task.**

**Examples**:

1. **Example**: Validated classification name input for adding classifications.  
   a. **File/Function**: views/inventory/add-classification.ejs, middleware/validateClassification.js  
   b. **Description**: Added client-side validation in add-classification.ejs using HTML5 pattern to block spaces and special characters in classification names. Server-side validation in validateClassification.js checks the same constraints, returning errors to the view if invalid.
2. **Example**: Implemented sticky forms with validation for inventory addition.  
   a. **File/Function**: views/inventory/add-inventory.ejs, middleware/validateInventory.js  
   b. **Description**: Designed add-inventory.ejs with sticky inputs to retain values on validation errors, using client-side checks for required fields like inv\_make and inv\_price. Server-side validation in validateInventory.js ensures data types and ranges, displaying errors in the view if validation fails.
3. **Example**: Validated account update and password change inputs.  
   a. **File/Function**: views/account/update.ejs, middleware/validateAccount.js  
   b. **Description**: Added client-side validation in update.ejs for email format and password requirements (e.g., length, characters). Server-side validation in validateAccount.js checks for unique email addresses and password strength, returning errors to the view for correction.

**Outcome 6: Demonstrate the skills of a productive team member (such as solving problems, collaborating with others, communicating clearly, fulfilling assignments, and meeting deadlines.)**

**Examples**:

1. **Example**: Resolved ReferenceError: errors is not defined in detail view.  
   a. **File/Function**: views/inventory/detail.ejs  
   b. **Description**: Identified and fixed a ReferenceError by removing unused errors check in detail.ejs, ensuring the /inv/detail/2 page, renders correctly. Collaborated with learning team via discussion boards to test and verify the fix, meeting assignment deadlines.
2. **Example**: Deployed and tested application on Render.com.  
   a. **File/Function**: GitHub repository, Render.com deployment  
   b. **Description**: Pushed the project to GitHub and deployed it to Render.com, thoroughly testing routes like /inv/detail/2 and /inv/type/1 in production. Communicated with team members to ensure all features (e.g., account management, inventory addition) worked, submitting URLs on time.