Assignment 3 Directions

Introduction

This week's activities saw us implementing dynamic code for the navigation bar, delivering inventory items by classification, become familiar with debugging, and implementing error handling. This assignment will expand each of these concepts, except for debugging, which is constant and ongoing.

Video Overview

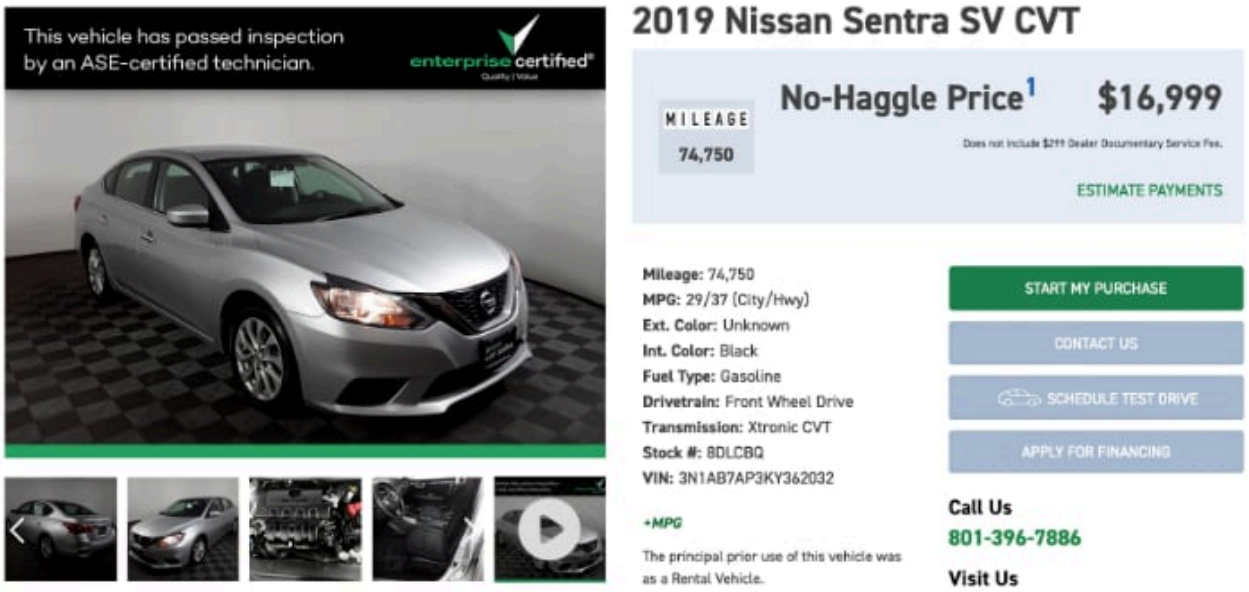
This short video demonstrates what the finished assignment may look like and the functionality it could possess. This is the Transcript of the video.

Tasks

Task One

1. Using the activities related to delivering the inventory items by classification view as an example, build a process to deliver a specific inventory item detail view. Note: This will be a **single view**, which will allow any vehicle's information to be displayed, based on the parameters passed through the URL. It is NOT a separate view for each item.
2. The detail delivery process must include:
    1. an appropriate route as part of the inventory route file,
    2. a controller function, which is part of the inventory controller,
    3. a function to retrieve the data for a specific vehicle in inventory, based on the inventory id (this should be a single function, not a separate one for each vehicle), which is part of the inventory-model,
    4. Build a custom function in the utilities > index.js file that will take the specific vehicle's information and wrap it up in HTML to deliver to the view,
    5. The view must be created and stored in the views > inventory folder and meet these specifications:
        - Display the vehicle make and model in the title element and in the main content heading.
        - Use the full-size image, not the thumbnail image.
        - The make, model, year and price should be prominent in the view. All descriptive data must also be displayed.

- The price must display as U.S. Dollars with appropriate currency symbol and proper place value commas.
- The mileage must display with proper place value commas.
- Each item must be displayed with appropriate supporting text to make sense to a site visitor. See this image from Enterprise car sales as an example:



- Meet the standards shown in the Frontend Checklist and display the vehicle information for easy reading and presenting a professional appearance.
- For large screens, the image and content must sit side-by-side (similar to the example car image). On small screens, the content should stack into a single column.
- The content (including images) must be responsive and maintain its clarity of presentation in the browser as the browser window grows or shrinks. No horizontal scrolling or zooming should be required, and the content should adapt to the screen and not overflow it.

Task 2

Apply the error handling middleware, from the activities, to all routes throughout the application. Make sure each process is included to catch and report errors should they occur in the application. You should test to make sure this works, but do NOT leave any intentional errors in place other than the one called for in task 3.

Task 3

Implement an intentional error process into the application (including a route, controller and middleware) that will generate a 500-type error. Add a link to trigger the route to the footer partial file. Make sure that when the link is clicked that the process is directed through a router and controller (these can be existing routers and controller or new just for this task), but the middleware catches the error and redirects to the error view. Ensure that the error view meets the requirements of the frontend checklist. See Throw exceptions in Node for a suggested means of causing the error to trigger.

Test, Test, Test

Be sure to complete all tasks and then test that they work.

1. Load the project to GitHub, then deploy to your render.com server.
2. Load the site into your browser.
3. Click a link in the main navigation bar.
4. From the classification view that should be presented in the browser click the link for any vehicle.
5. The details should be visible in a new view. The details are clear, professional in appearance that can be easily consumed by the human looking at the browser.
6. Change the screen size to emulate delivery to various device sizes. The content should be responsive and adjust to allow clear presentation regardless of screen size.
7. Check the view for HTML and CSS validity and WAVE compliance.

8. Repeat multiple times (clicking on navigation link, then "drilling down" to the detail) to ensure that the process works with all vehicles.
9. Test for a non-existent view using a fake route. A 404 error should be delivered via the error view.
10. Click the error link in the footer. A 500 error should be delivered via the error view.

Submission

- Work with your learning team to accomplish the tasks and post to and read the weekly discussion board to help one another.
- Build, run and test to make sure that each of the tasks works.
- When satisfied that the code is operational:
    1. Upload the project to your GitHub repository.
    2. Go to the render.com dashboard and manually deploy to your web service.
    3. Thoroughly test the application in the browser on the production server following deployment.
    4. Fix any and all errors prior to submission.
    5. Submit the render.com production URL **AND** the GitHub repository URL as a comment in the assignment submission.

Grading Matrix

The operation of this process should follow the principles from the class. This assignment has values in multiple objectives as shown below:

Objective 1

- The detail view meets the standards shown in the Frontend Checklist.
- The detail information is responsive and can be easily read on changing screen sizes without zooming or horizontal scrolling. This includes a multi-column layout on large screens and stacked layout on small screens.
- The price is formatted correctly using commas and contains the currency symbol for US dollars.
- The mileage is formatted correctly and uses commas in the appropriate place values.
- Obj. 1 value: Refer to grade book

Objective 2

- A route exists to handle the incoming detail request and works correctly.
- The inventory controller contains the control structure and process to deliver the detail view and data and works correctly.
- A new custom function exists in the utilities > index.js file to build HTML around the vehicle detail data, then return the resulting string to the controller and works correctly.
- Obj. 2 value: Refer to grade book

Objective 3

- The detail delivery process works and is an MVC solution, including route, controller, model and utility as required.
- The footer-based error process works and uses an MVC approach.
- Obj. 3 value: Refer to grade book

Objective 4

- A function exists in the inventory model to get data for a particular inventory item based on the inventory id, uses a Parameterized Statement approach, and works.
- Obj. 4 value: Refer to grade book

Objective 5

- Error handling has been implemented throughout the routes and work successfully to deliver error views when an error is detected.

- The footer-based error link is present and works generate an error, as described in the directions, which is caught and handled using the error handler middleware.
- Obj. 5 value: Refer to grade book

Objective 6

- The render.com production URL and the GitHub repository URL are both submitted on time and correctly. Both URL's must be present and operational for the assignment to be graded.
- Obj. 6 value: Refer to grade book