Vehicle Registration Form Project Instructions

1 PROJECT ABSTRACT

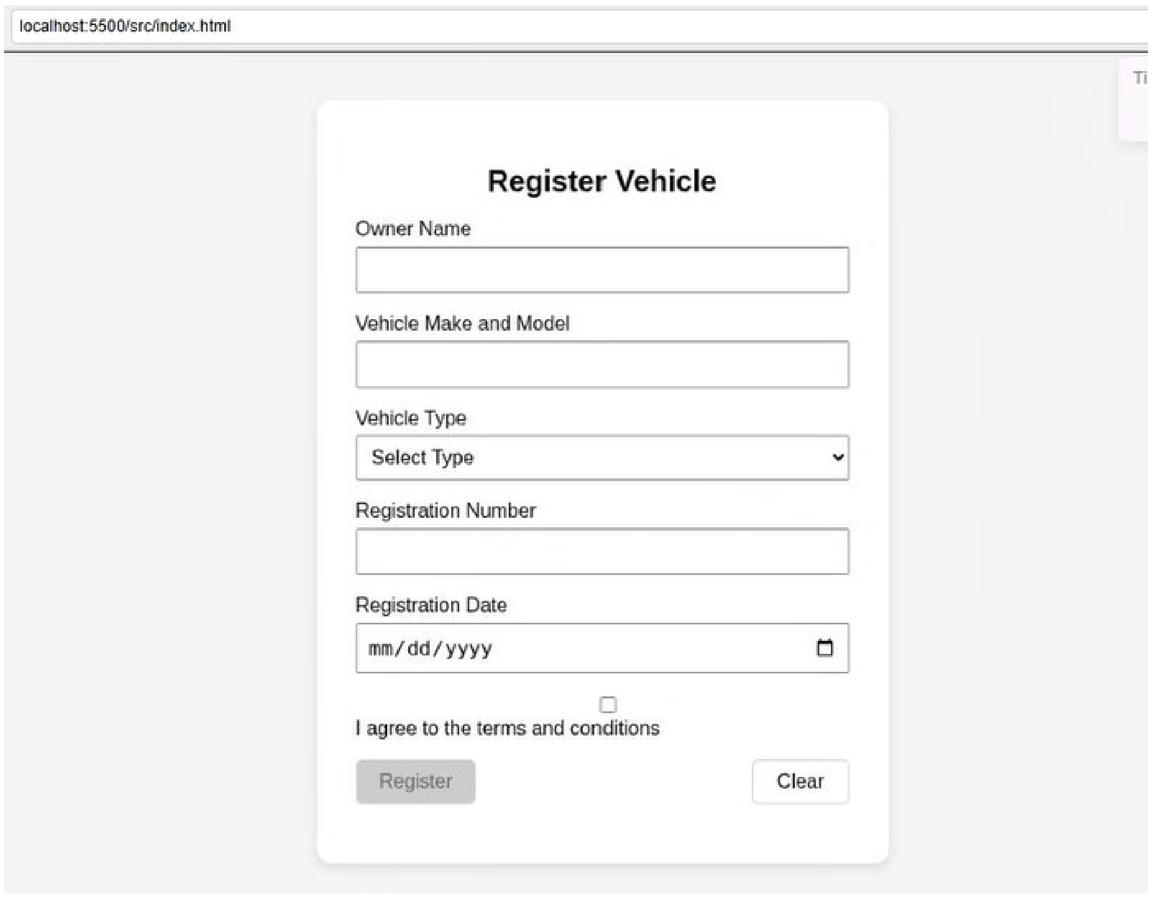
The **Vehicle Registration Form** is a front-end web project focused on building an user-friendly form using **HTML**, **CSS**, and **TypeScript**. This project allows users to register a vehicle by providing details such as the owner's name, vehicle make and model, type, registration number, and registration date, with built-in real-time validation and error feedback.

This project emphasizes:

- Validating user inputs like owner name, make and model, registration number, and date using custom TypeScript logic.
- Ensuring required fields are filled before enabling form submission, with live feedback and input highlighting for invalid entries.
- Displaying appropriate error messages and guiding the user to correct mistakes before submitting the form.
- Implementing a clean form layout using structured **HTML** and modern **CSS**.
- Managing form state dynamically, including enabling/disabling the **Register** button and resetting the form using a **Clear** button.

ScreenShots:

*****Register Page*****



localhost:5500/src/index.html

Register Vehicle

Owner Name

Vehicle Make and Model

Vehicle Type

Select Type

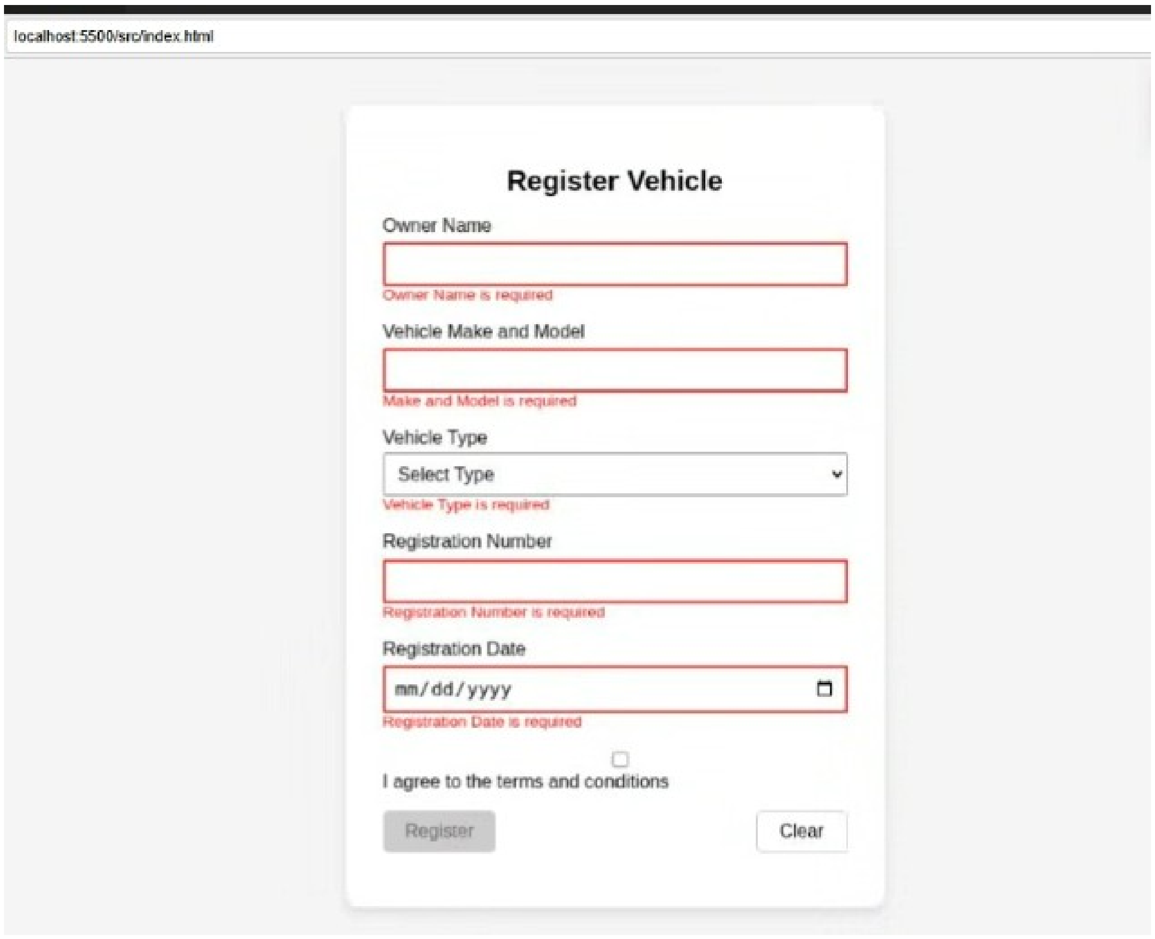
Registration Number

Registration Date

mm/dd/yyyy

I agree to the terms and conditions

Validations


localhost:5500/src/index.html

Register Vehicle

Owner Name

Owner Name is required

Vehicle Make and Model

Make and Model is required

Vehicle Type

Select Type

Vehicle Type is required

Registration Number

Registration Number is required

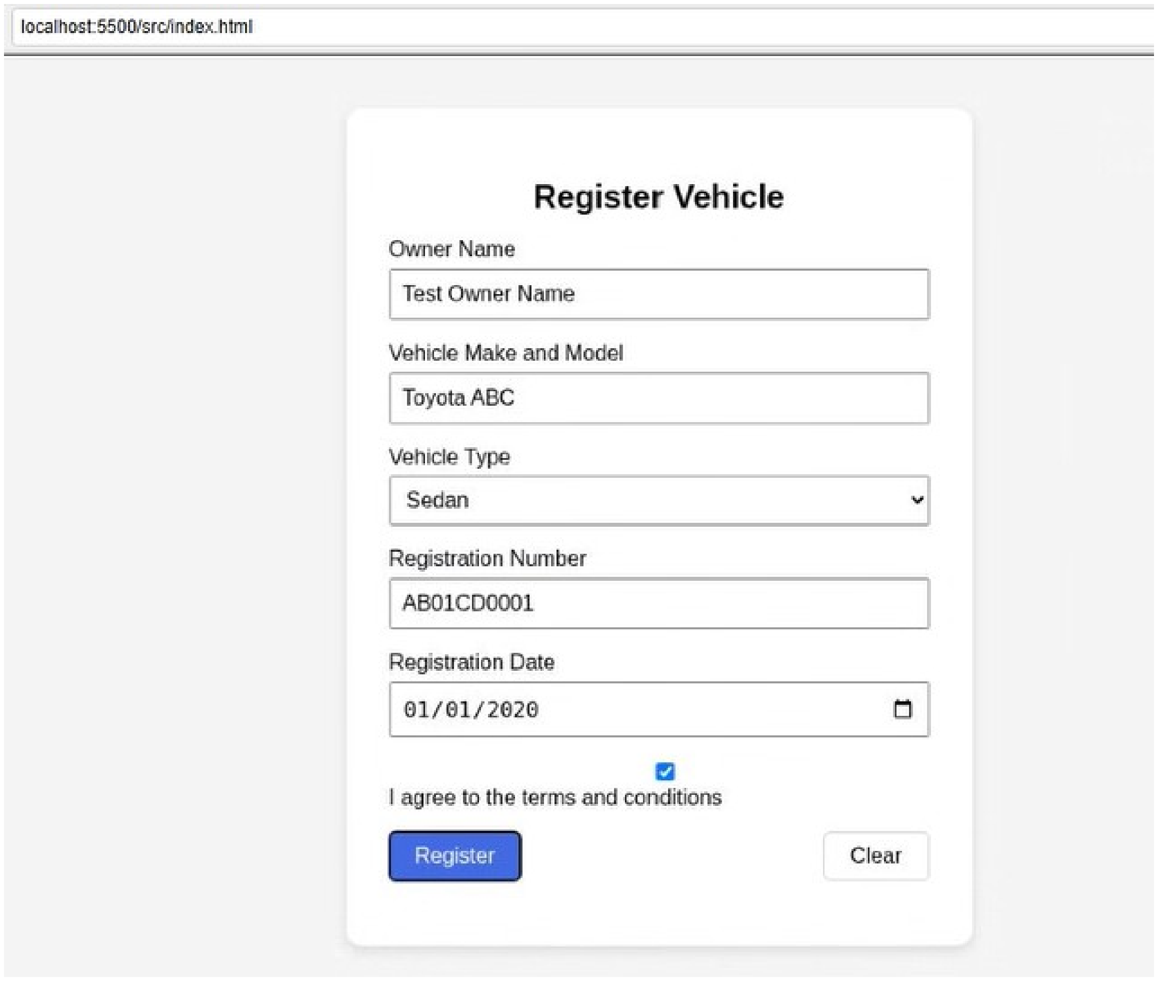
Registration Date

mm/dd/yyyy

Registration Date is required

I agree to the terms and conditions

*****All required fields are correctly filled, Additionally, the terms and conditions checkbox is checked, which is essential for enabling the Register button*****


localhost:5500/src/index.html

Register Vehicle

Owner Name: Test Owner Name

Vehicle Make and Model: Toyota ABC

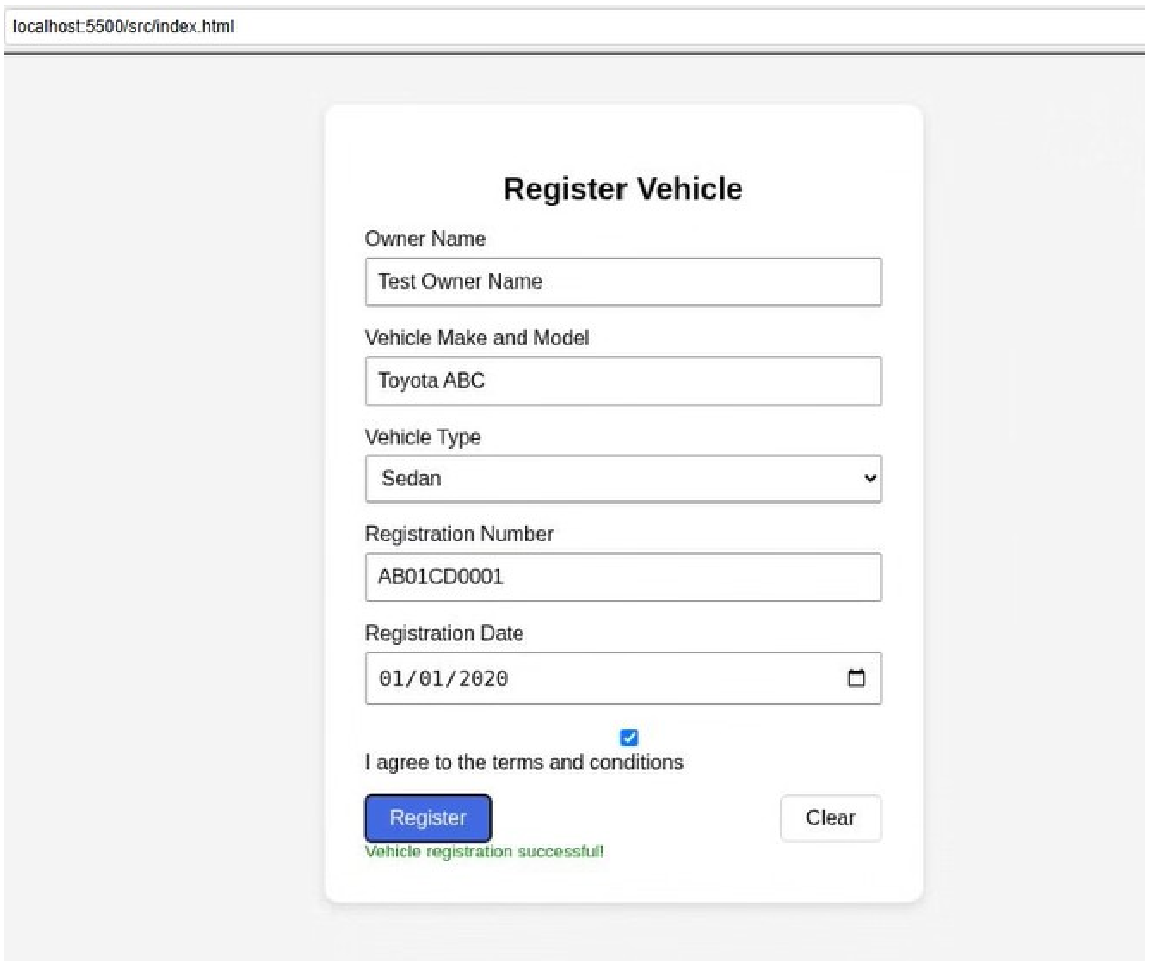
Vehicle Type: Sedan

Registration Number: AB01CD0001

Registration Date: 01/01/2020

[✓] I agree to the terms and conditions

Register Successful



localhost:5500/src/index.html

Register Vehicle

Owner Name: Test Owner Name

Vehicle Make and Model: Toyota ABC

Vehicle Type: Sedan

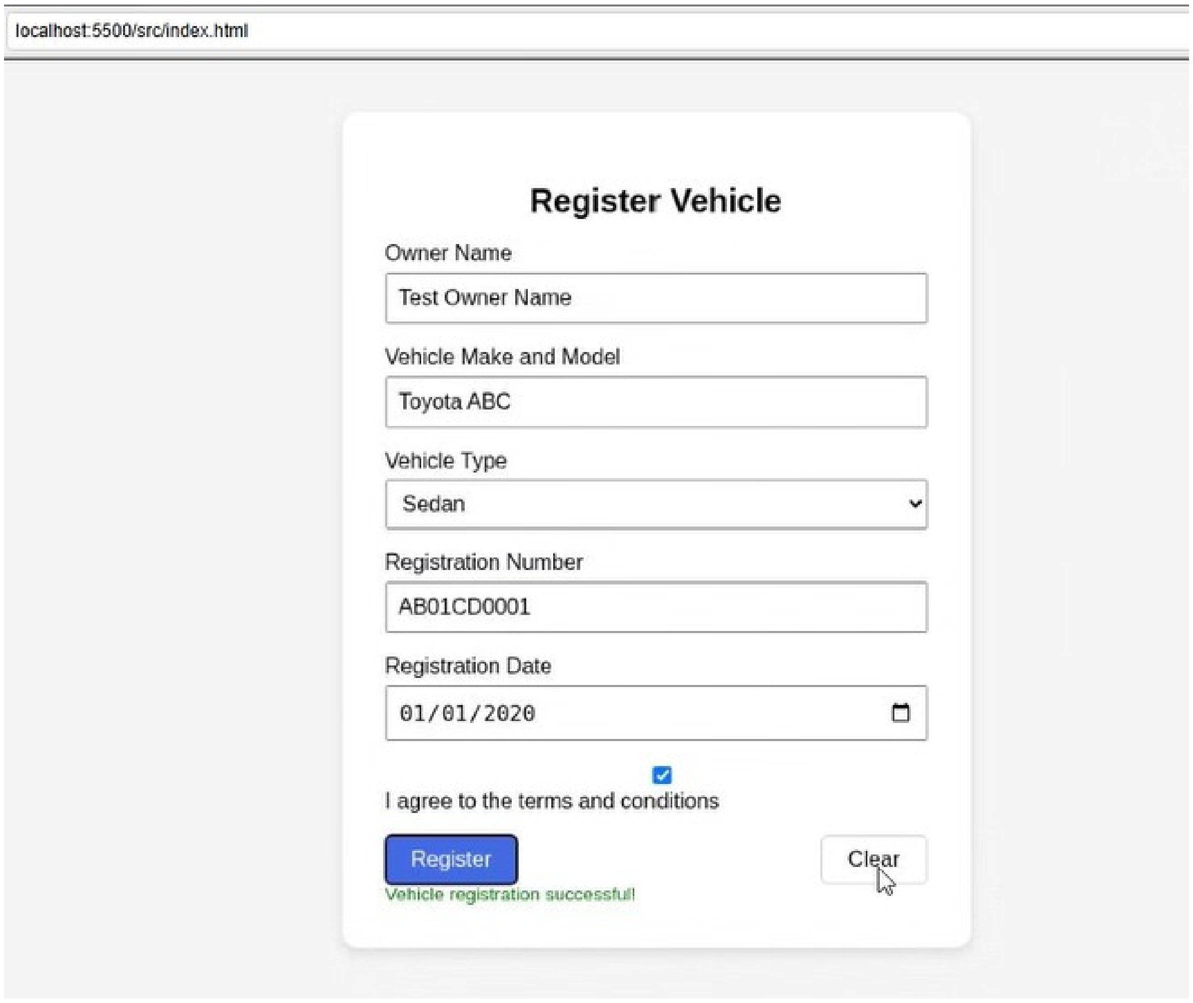
Registration Number: AB01CD0001

Registration Date: 01/01/2020

[✓] I agree to the terms and conditions

Vehicle registration successful!

*****Clicking the Clear button resets the entire form — all input fields are emptied, validations and highlights are removed, and the Register button is disabled again. This ensures a clean state for the user to start a fresh registration.*****


localhost:5500/src/index.html

Register Vehicle

Owner Name: Test Owner Name

Vehicle Make and Model: Toyota ABC

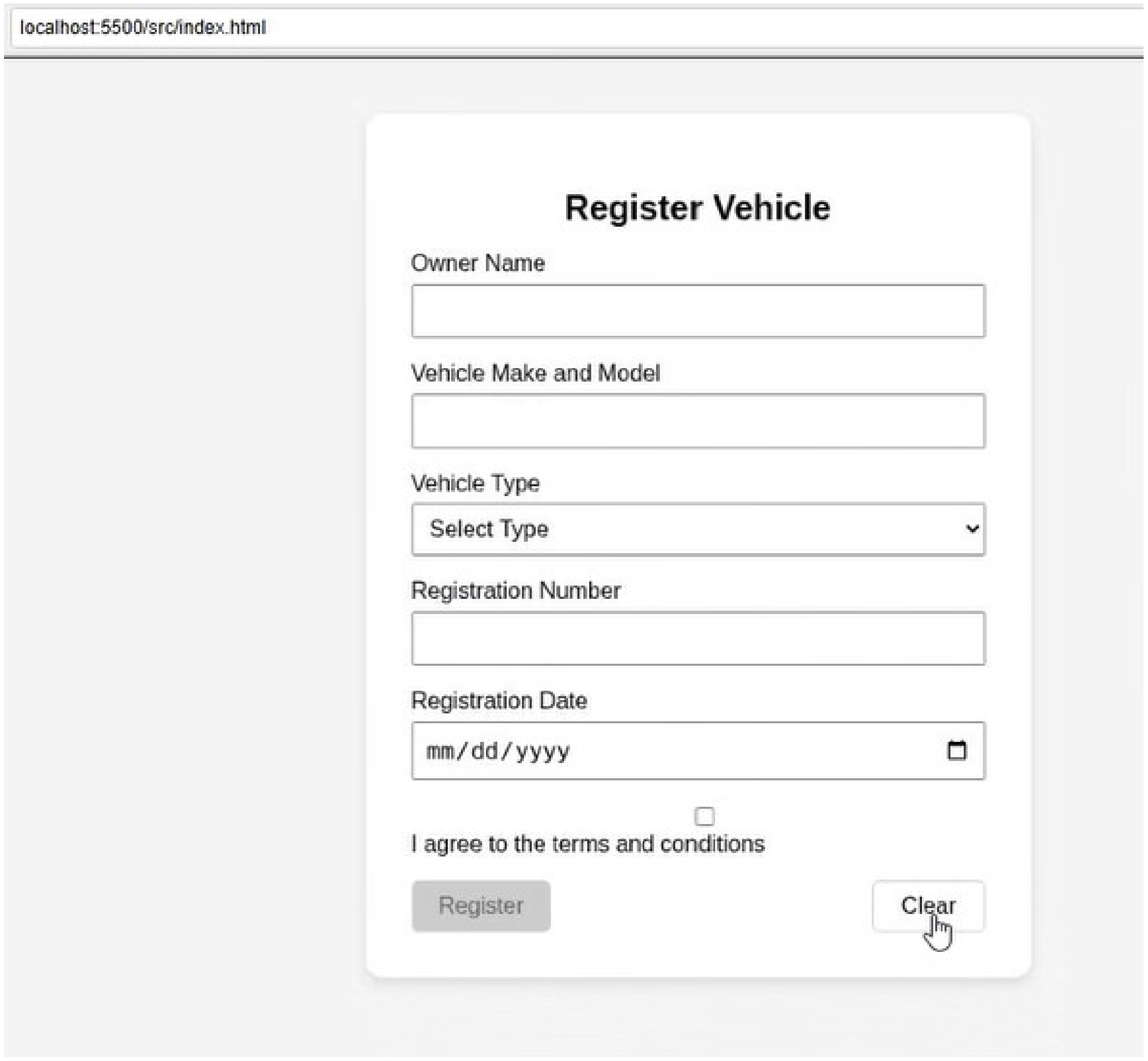
Vehicle Type: Sedan

Registration Number: AB01CD0001

Registration Date: 01/01/2020

☑ I agree to the terms and conditions

Vehicle registration successful!


localhost:5500/src/index.html

Register Vehicle

Owner Name

Vehicle Make and Model

Vehicle Type

Select Type

Registration Number

Registration Date

mm/dd/yyyy

☐ I agree to the terms and conditions

Instructions for Core Files

1. index.html

Create the HTML structure for a **Vehicle Registration Form** using the following specifications:

- In the `head`, include:
    - Title as **“Register Vehicle”**
    - Link to external `style.css` for styling
- In the `body`, add a `form` with ID `registerForm`.
- Inside the form:
    - Add a heading `h2` with the text "**Register Vehicle**"
    - Add input fields for:
        - **Owner Name**: Field with ID `ownerName` for entering the vehicle owner's name; it should be required.
        - **Vehicle Make and Model**: Field with ID `makeModel` for entering vehicle details; it should be required.
        - **Vehicle Type**: Dropdown field with ID `vehicleType`; options include Sedan, SUV, Truck, and Motorcycle. It should be required.
        - **Registration Number**: Field with ID `regNumber` to enter the vehicle's registration number; it should be required.
        - **Registration Date**: Date input field with ID `regDate` to select the date of registration; it should be required.
    - Include a checkbox with ID `terms` for agreeing to terms and conditions; this must be checked to allow submission.
    - Add two buttons:
        - **Register Button**: Submit button with ID `submitBtn`, initially disabled and only enabled when all required fields are filled correctly.
        - **Clear Button**: Button that resets all fields and clears validations.
    - Include an area with ID `outputMessage` to display form messages (success or error).
- Link the external JavaScript file: `script.js` at the end of the body.

- At the end of the body (just before closing it), include a script tag that loads the external dynamically built JavaScript file named `script.js`.

File path: “../dist/script.js”

2. style.css

Define the styles for the Vehicle Registration Form with the following specifications:

- Use a clean, readable font and center the form layout on the page.
- Style the form with:
  - Padding
  - Rounded corners
  - Subtle box-shadow for a card-like appearance
- Apply consistent spacing and styling for form groups and inputs.
- Style the following elements:
  - `.error` for red-colored error messages
  - `.success` for green-colored success messages
  - `.highlight-error` to highlight invalid inputs with a red border
  - `.button-row` to space out the action buttons
- Customize buttons:
  - **Register button**: Blue background with white text
  - **Clear button**: White background with light border
  - **Disabled state**: Greyed out appearance with disabled pointer and interaction

---

3. script.ts

Write the TypeScript logic to handle all form interactivity and validations:

- Define validation rules:
  - All fields except the vehicle type must not be empty
  - Vehicle type must be selected (not left as the default "Select Type")
  - The terms checkbox must be checked before allowing submission
- Functions to implement:
  - `validateField(field, message)` – checks if a field has a valid value and displays error messages
  - `validateTerms()` – verifies if the terms checkbox is selected
  - `isFormValid(force)` – validates all required fields and returns overall form validity
  - `updateSubmitState()` – enables or disables the Register button based on validation status
  - `validateOnSubmit()` – displays final success or error message when form is submitted
  - `clearForm()` – resets all form fields, clears errors, and disables the submit button
  - `setupRealTimeValidation()` – adds input and blur event listeners to perform live validation
- Initialize the form on page load, set up all event listeners, and ensure the form is non-submittable until all validations pass.

Detailed HTML Structure Guide for Vehicle Registration Form App

1. Document Declaration and Head Setup

Begins with the HTML5 doctype declaration and sets up page metadata:

- Inside the `head` section:
  - Sets the page `title` to “Register Vehicle”.
  - Links an external stylesheet via `link` pointing to `style.css`.

2. Form Container and Page Body

Wraps the registration form inside a `form` element:

- Uses a `form` element with the `id registerForm` to group all fields.
- Place the form inside the `body` of the HTML page.

3. Form Title

Displays a heading above the form:

- Adds a level-two heading with the text “Register Vehicle”.

4. Owner Name Input

Captures the name of the vehicle owner:

- Adds a `label` with the text “Owner Name”.
- Includes a text input field with `id ownerName` and `required` attribute.
- Adds a `div` with `id ownerNameError` and class `error` to display validation messages.

5. Make and Model Input

Allows entry of the vehicle make and model:

- Uses a `label` with the text "Vehicle Make and Model".
- Adds a text input with `id makeModel`.
- Includes a `div` with `id makeModelError` for validation errors.

6. Vehicle Type Dropdown

Provides vehicle category options:

- Adds a `label` with the text "Vehicle Type".
- Uses a `select` element with `id vehicleType`.
- Includes four dropdown `option` values: Sedan, SUV, Truck, Motorcycle.
- Includes a default option `Select Type` with an empty value.
- Adds a `div` with `id vehicleTypeError` for error feedback.

7. Registration Number Input

Takes in the unique vehicle registration number:

- Uses a `label` with the text "Registration Number".
- Includes a text input with `id regNumber`.
- Provides a corresponding `div` with `id regNumberError` for validation messages.

8. Registration Date Input

Collects the registration date of the vehicle:

- Adds a `label` with the text "Registration Date".
- Uses a `date` input field with `id regDate`.
- Includes an `error` message `div` with `id regDateError`.

9. Terms and Conditions Checkbox

Ensures user agrees to the conditions before submission:

- Adds a checkbox input with `id terms`.
- Accompanied by a `label` that says "I agree to the terms and conditions".
- Displays errors in a `div` with `id termsError` if the checkbox isn't ticked.

10. Form Buttons Row

Displays the action buttons:

- Wrapped in a `div` with class `button-row`.
- First button is the **Submit** button with `id submitBtn` and `disabled` attribute initially set.
- Second button is a **Clear** button that calls `clearForm()` on click.

11. Output Section

Shows the final message after validation:

- Includes a `div` with class `output` and `id outputMessage` below the buttons.
- Will display either a success message or error message based on form validation.

12. JavaScript File Link

Includes interactivity by linking the JavaScript file:

- Link the external JavaScript file: script.js at the end of the body.
- At the end of the body (just before closing it), include a script tag that loads the external dynamically built JavaScript file named script.js.
- File path: "../dist/script.js"
- Ensures script loads **after** all HTML elements have been rendered.

Detailed CSS Styling Guide for Registration Form App

This guide explains how the `style.css` file styles the vehicle registration form. Each section describes how styles improve visual design and usability.

1. Body Styling

Applies base styling and centers the form:

- **Font**: Uses `Arial, sans-serif` for clean, legible text.
- **Background color**: `#f4f4f4` (a soft light gray for subtle contrast).
- **Display**: Sets to `flex` to center content horizontally.
- **Horizontal alignment**: `justify-content: center` centers the form.

- **Padding**: Adds `2rem` of space around the edges to prevent content crowding.

2. Form Container (form)

Creates a card-like appearance for the form:

- **Background**: `white` for a clean, paper-like look.
- **Padding**: `2rem` inside the form for content spacing.
- **Border-radius**: `12px` for smooth, rounded corners.
- **Width**: `400px` fixed width to keep layout tidy and readable.
- **Box-shadow**: `0 4px 8px rgba(0, 0, 0, 0.1)` for subtle elevation effect.

3. Form Heading (h2)

Styles the form title:

- **Text alignment**: `center` for clear focus.
- **Bottom margin**: `1rem` to separate the heading from the first input.

4. Form Group Wrapper (.form-group)

Spaces out input sections uniformly:

- **Bottom margin**: `1rem` adds vertical spacing between field sections.

5. Form Labels (label)

Aligns and styles field labels clearly:

- **Display**: `block` ensures each label appears on its own line.
- **Bottom margin**: `0.3rem` for a small gap between label and input.

6. Input Fields and Select Dropdowns (input, select)

Provides consistent input field appearance:

- **Width**: `100%` to span full container width.
- **Padding**: `0.5rem` for a comfortable typing area.
- **Font size**: `1rem` for legible input text.

- **Box-sizing**: `border-box` ensures padding doesn't overflow container.

7. Error Message Styling (.error)

Formats validation error messages:

- **Text color**: `red` to indicate issues.
- **Font size**: `0.85rem` for subtle but visible messaging.

8. Success Message Styling (.success)

Displays validation success messages:

- **Text color**: `green` to signal valid state.
- **Font size**: `0.85rem` for consistency with error messages.

9. Output Message Section (.output)

Shows the result of the form submission:

- **Text alignment**: `center` for symmetry.
- **Top margin**: `1rem` to separate it from form buttons.
- **Font weight**: `bold` to draw attention.

10. Input Error Highlighting (input.highlight-error)

Visually emphasizes incorrect fields:

- **Border**: `2px solid red` to clearly flag invalid inputs.

11. Button Row Layout (.button-row)

Aligns Submit and Clear buttons:

- **Display**: `flex` to lay out buttons horizontally.
- **Justify content**: `space-between` evenly spaces the buttons.
- **Top margin**: `1rem` to push buttons below input area.

12. Button Styling (button)

Gives all buttons a unified appearance:

- **Padding**: `0.5rem 1.2rem` for a balanced clickable area.
- **Font size**: `1rem` for consistency.
- **Cursor**: `pointer` to signal interactivity.
- **Border**: `none` to remove default outlines.
- **Border-radius**: `5px` for smooth corners.

13. Submit Button (button[type="submit"])

Emphasizes the primary action button:

- **Background color**: `royalblue` for strong visibility.
- **Text color**: `white` for clear contrast.

14. Clear Button (button[type="button"])

Creates a lighter secondary action:

- **Background color**: `white` to appear non-primary.
- **Border**: `1px solid #ccc` for soft framing.

15. Disabled Button Styling (button:disabled)

Indicates inactive buttons clearly:

- **Background color**: `#ccc !important` (gray for inactive look).
- **Text color**: `#666 !important` for dimmed appearance.
- **Cursor**: `not-allowed !important` to block interaction.
- **Note**: `!important` ensures styles override other states.

Detailed TypeScript Logic Guide for Registration Form App

This guide explains the behavior of `script.ts`, describing each functional block and how it contributes to validating and managing the vehicle registration form.

1. Error Display Functions

`showFieldError(field, message)`

Utility Method

- **Role:** Displays the given error message and highlights the invalid field with a red border.
- **Called by:** `validateField()` when a validation condition fails.
- Get the corresponding error element using the field's ID (e.g., `"regNumber"` → `"regNumberError"`).
- Set its text content to the error message.
- Add the class `highlight-error` to the field.

`clearFieldError(field)`

Utility Method

- **Role:** Removes error message and resets field styling.
- **Called by:** `validateField()` when a field becomes valid.
- Clear the related error element.
- Remove `highlight-error` from the field.

2. Field Validation Function: `validateField(field, message, force)`

Validation Trigger

Validates individual form fields with custom messages:

- **Role:** Checks if a specific field has valid input.
- **Linked with:** Real-time validation events (input, blur).
- **Force Parameter:** Skips the `touchedFields` check when set to `true` (used during final submission).
- Check if the field's value is empty.
- If empty → call `showFieldError()`.
- If filled → call `clearFieldError()`.

3. Terms and Conditions Validation

`validateTerms(force)`

Checkbox Validation

- **Role:** Validates if the "I agree to terms" checkbox is checked.
- **Used in:** Real-time checkbox change and on submission.
- If unchecked → show `"You must accept terms"` inside `termsError`.
- If checked → clear the error message.

`showTermsError()`

Utility Method

- **Role:** Displays an error if the terms checkbox is not accepted (on form submit).
- **Used in:** `isFormValid()` when forcing validation.
- Set error message: `"You must accept terms"` in the terms error span.

4. Form Validity Check: `isFormValid(force)`

Main Validation Aggregator

Evaluates the overall form state:

- **Role:** Validates all fields in the form.
- **Returns:** `true` if all fields and checkbox are valid.
- **Force param:** Triggers full validation regardless of touch state.
- Loop through the fields below and apply validation **with the following specific messages**:

Field ID	**Validation Rule**	**Error Message**
ownerName	Must not be empty	"Owner Name is required"
makeModel	Must not be empty	"Make and Model is required"
vehicleType	Must be selected (not empty string)	"Vehicle Type is required"
regNumber	Must not be empty	"Registration Number is required"

regDate	Must not be empty	"Registration Date is

		required"
`terms` checkbox	Must be checked	"You must accept terms"

- Loop through each field (use `touchedFields[field.id]` to determine if it's been interacted with unless `force` is true).
- If invalid:
  - Call `showFieldError(field, message)`
- If valid:
  - Call `clearFieldError(field)`
- Finally, ensure `terms` checkbox is validated via `validateTerms()` or `showTermsError()`.
- Return `true` only if all fields are valid and terms are checked.

5. Submit Button Enable/Disable: `updateSubmitState()`

Button Enable Logic

Enables or disables the "Register" button:

- **Role:** Enables or disables the "Register" button.
- **Used in:** Every `input`, `blur`, and `change` event.
- Check if **all fields are filled**.
- Call `isFormValid(false)` to ensure validity.
- Enable the submit button only if everything is valid.

6. Form Submission Validation: `validateOnSubmit()`

Final Submission Handler

Handles submit button click:

- **Role:** Called when the user clicks the Register button.
- **Force-validates** all fields using `isFormValid(true)`.
- Show success message (`"Vehicle registration successful!"`) in green if valid.
- Else, show `"Please fix the errors above."` in red.
- Apply class `success` or `error` to `outputMessage`.

7. Form Reset Logic: `clearForm()`

Form Reset Utility

Resets form to initial clean state:

- **Role:** Clears all input fields, error messages, and resets the form.
- **Used in:** Clear button's `onclick`.
- Removes all validation messages and styling.
- Resets the `touchedFields` tracker to false for every field.

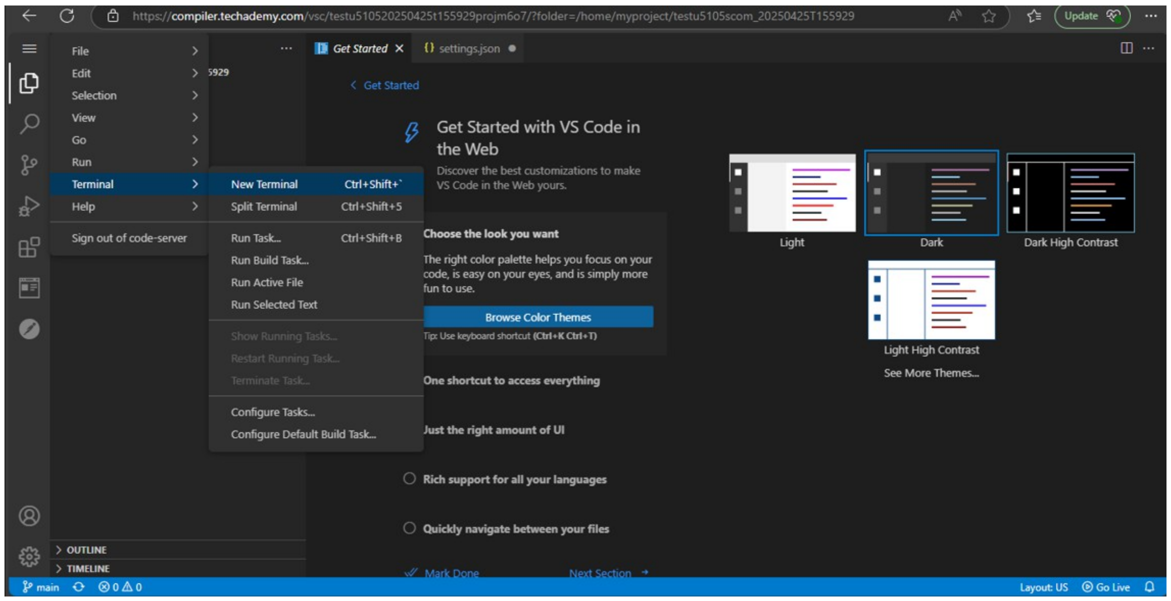
Assessment Guidelines

Step 1:

- Once the VS Code interface loads in the browser, wait until you see the workspace and left sidebar.
- To open the command terminal the test takers, need to go to

  Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.

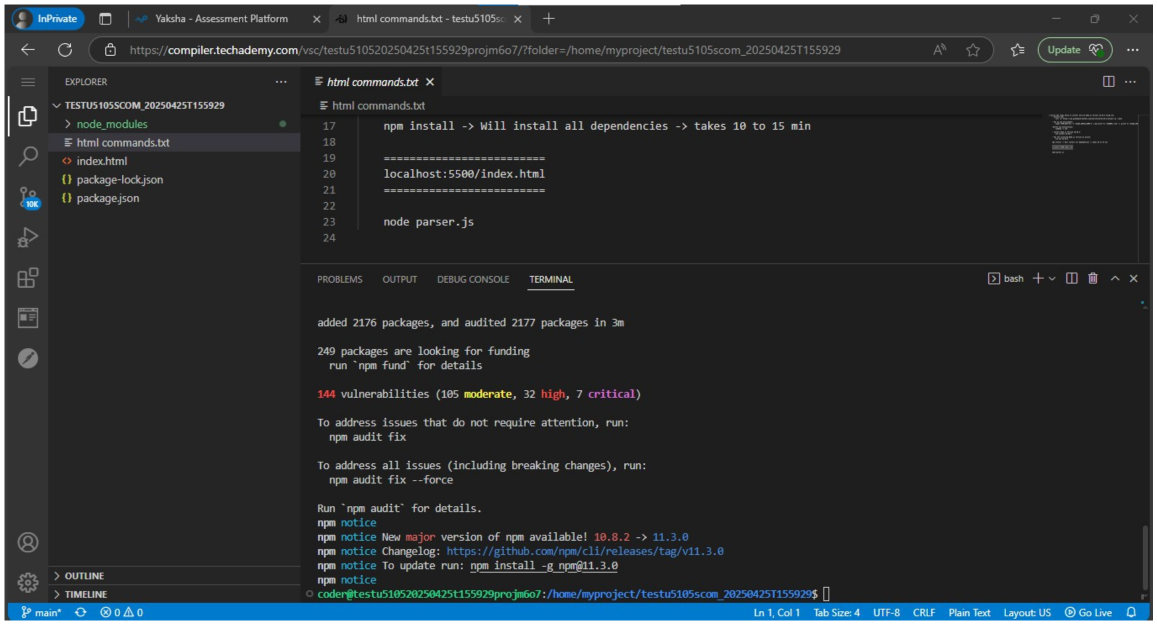
Now in the terminal you need to install all dependencies using the **"npm install"** command.


https://compiler.techademy.com/vsc/testu510520250425t155929projm6o7/?folder=/home/myproject/testu5105scom_20250425T155929
Update
File
Edit
Selection
View
Go
Run
Terminal
Help
Sign out of code-server
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task... Ctrl+Shift+B
Run Build Task...
Run Active File
Run Selected Text
Show Running Tasks...
Restart Running Task...
Terminate Task...
Configure Tasks...
Configure Default Build Task...
Get Started
settings.json
Get Started
Get Started with VS Code in the Web
Discover the best customizations to make VS Code in the Web yours.
Choose the look you want
The right color palette helps you focus on your code, is easy on your eyes, and is simply more fun to use.
Browse Color Themes
Tip: Use keyboard shortcut (Ctrl+K Ctrl+T)
One shortcut to access everything
Just the right amount of UI
Rich support for all your languages
Quickly navigate between your files
Mark Done
Next Section
Light
Dark
Dark High Contrast
Light High Contrast
See More Themes...
OUTLINE
TIMELINE
main
Layout: US
Go Live

Step 2:

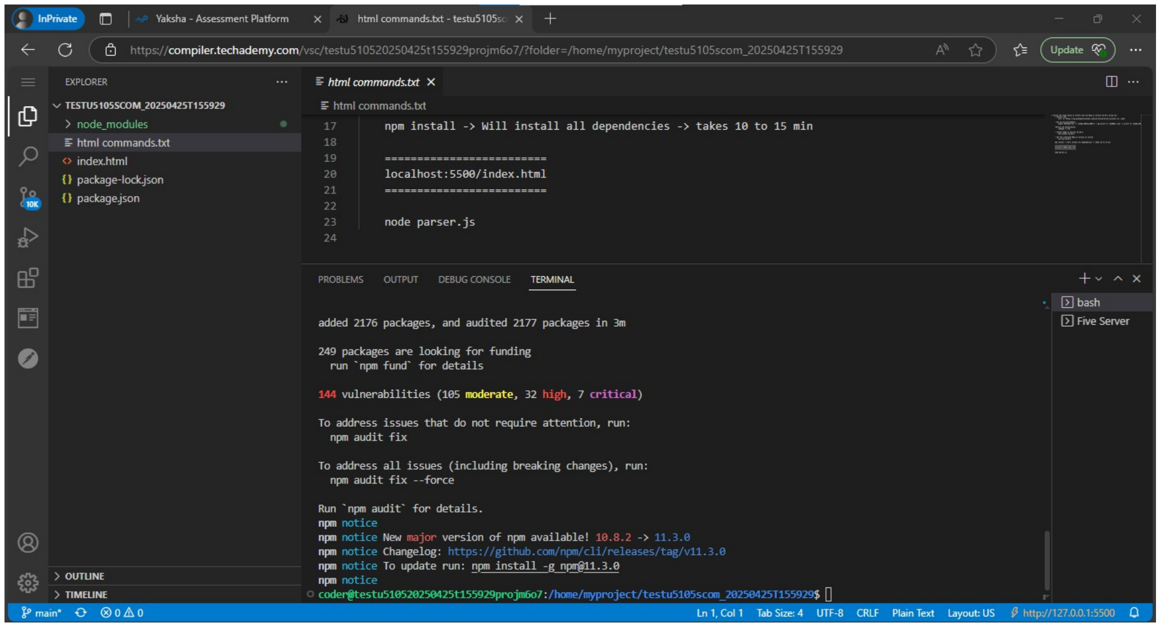
- Once installation completes, go to the **bottom right corner** of the VS Code screen.

- **Make sure you run 'npm build run' command from your terminal to generate the JavaScript file from Typescript before running the application, after every change in Typescript file**
- Click the "**Go Live**" button – This will start a **live server**, The server will run at port `5500` (e.g., `http://localhost:5500/`)



```
npm install -> Will install all dependencies -> takes 10 to 15 min

========================
localhost:5500/index.html
========================

node parser.js
```

```
added 2176 packages, and audited 2177 packages in 3m

249 packages are looking for funding
  run `npm fund` for details

144 vulnerabilities (105 moderate, 32 high, 7 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
npm notice
npm notice New major version of npm available! 10.8.2 -> 11.3.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.3.0
npm notice To update run: npm install -g npm@11.3.0
npm notice
coder@testu510520250425t155929projm6o7:/home/myproject/testu5105scom_20250425T155929$
```



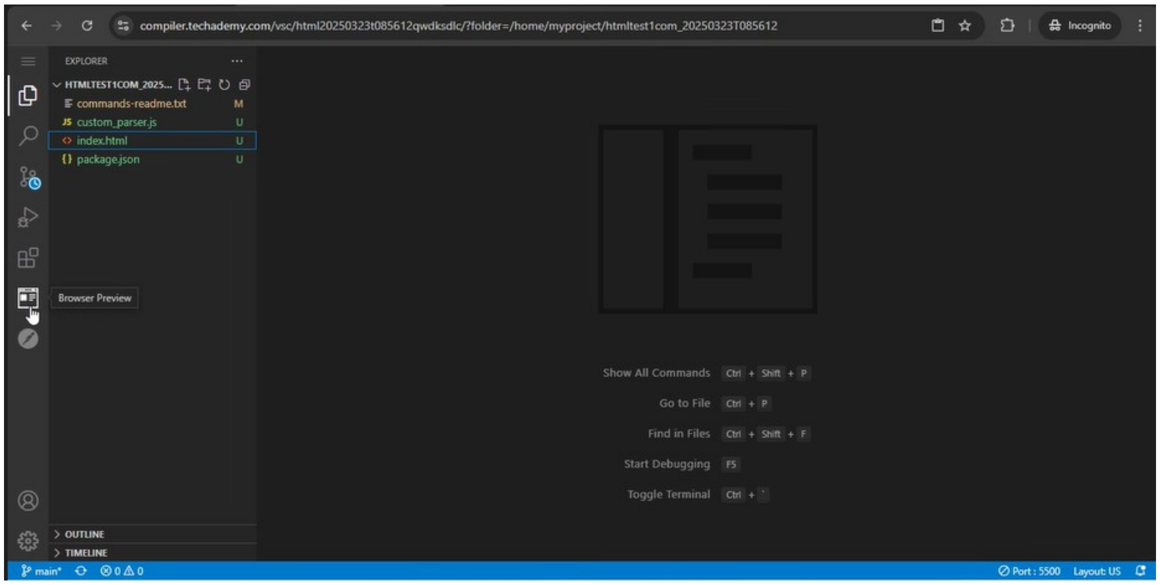
```
npm install -> Will install all dependencies -> takes 10 to 15 min

========================
localhost:5500/index.html
========================

node parser.js
```

```
added 2176 packages, and audited 2177 packages in 3m

249 packages are looking for funding
  run `npm fund` for details

144 vulnerabilities (105 moderate, 32 high, 7 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
npm notice
npm notice New major version of npm available! 10.8.2 -> 11.3.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.3.0
npm notice To update run: npm install -g npm@11.3.0
npm notice
coder@testu510520250425t155929projm6o7:/home/myproject/testu5105scom_20250425T155929$
```

Step 3: Preview Output in Browser

- This is a **web-based application**, so to view it in a browser, use the **internal browser inside the workspace**.

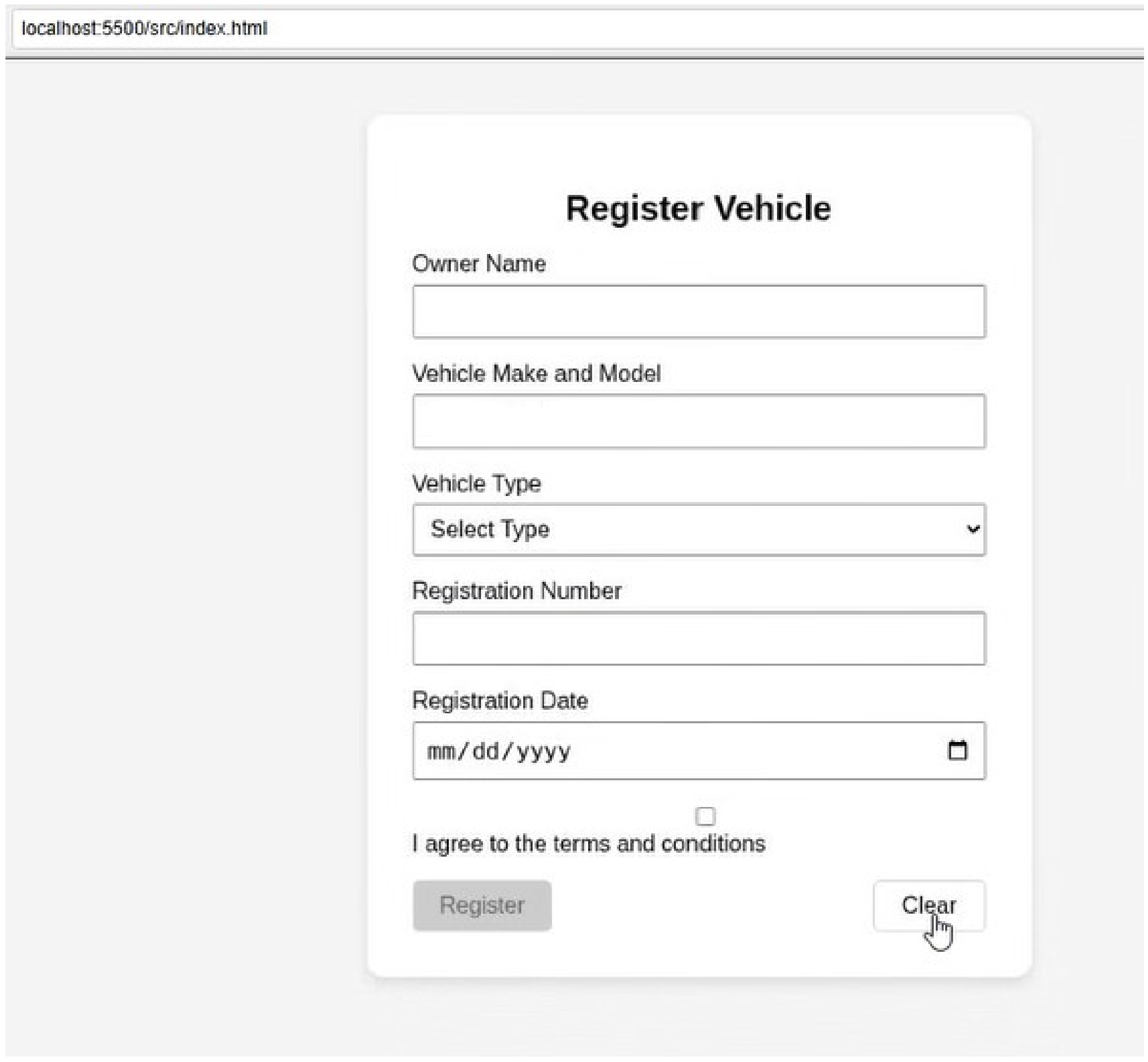
- Click on the **second last icon on the left panel** (the one labeled **"Browser Preview"**). This will open a tab within VS Code where you can **launch and view your application**.
- Note: The application **will not open in your system's local browser** — it must be viewed using the internal browser.



In the **Browser Preview tab**, type the following URL in the address bar and press **Enter**:

Your file is being served on: localhost:5500/src/index.html

This will load your HTML file and display the output of your web page **inside the internal browser**.


localhost:5500/src/index.html

Step 4:

- Go back to the **terminal** and type the following command, then press **Enter**:

  node src/test/custom-parser.js

- This command will **execute the validation script** and display the test results for your HTML file in the terminal.

Mandatory Assessment Guidelines:

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.**
3. **This editor Auto Saves the code.**
4. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
5. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

   Note: The application will not run in the local browser
6. **You can follow series of command to setup environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min.**
   b. **npm run build -> Will transpile your Typescript file to Javascript**
   c. **node src/test/custom-parser.js -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**
7. **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.**