Bill Splitter Project Instructions

1 Project Abstract

The Bill Splitter App is a practical web development project designed to strengthen core front-end skills using **HTML**, **CSS**, and **TypeScript**. The goal of this project is to create an interactive application that calculates how a total bill, including a tip, can be split evenly among a group of people.

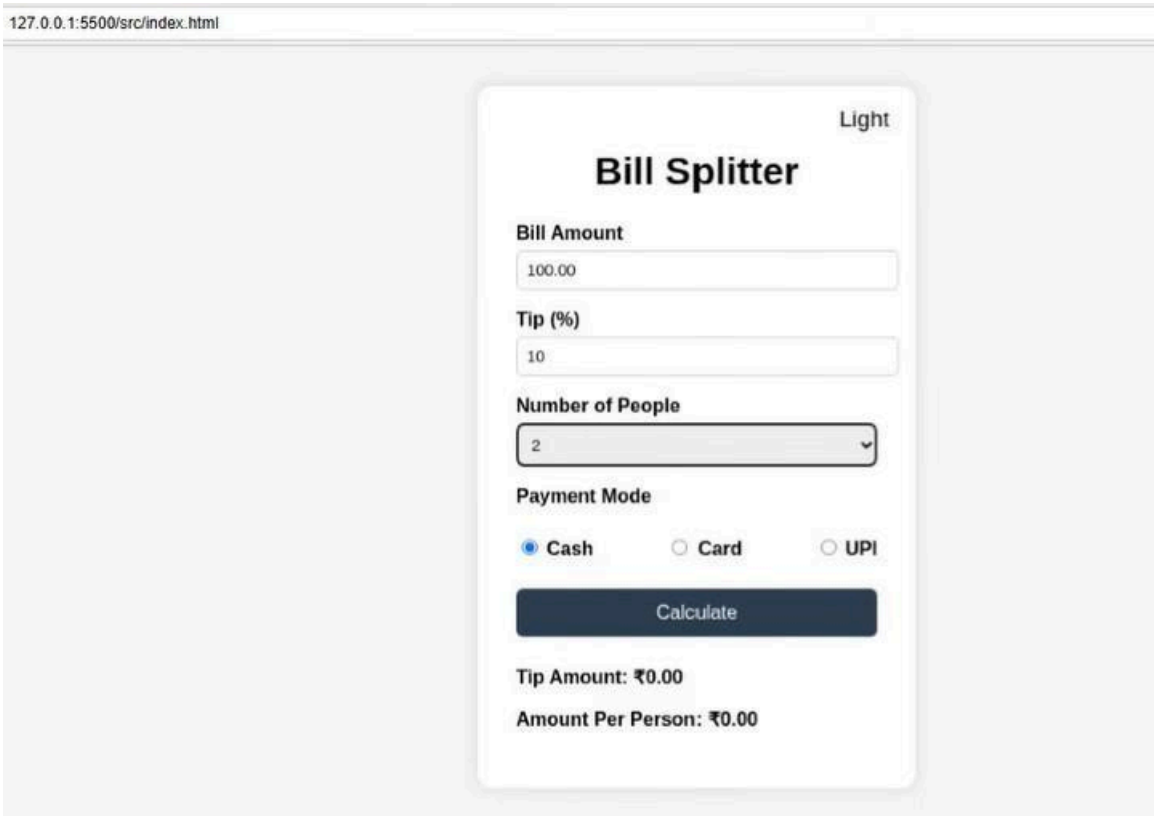
You will build a clean, user-friendly interface where users can enter the bill amount, tip percentage, and number of people sharing the bill. The app should dynamically calculate the **tip amount** and **amount per person**, displaying results instantly upon clicking the "Calculate" button. Additionally, users can be able toggle between light and dark themes to enhance usability and accessibility.

This project emphasizes:

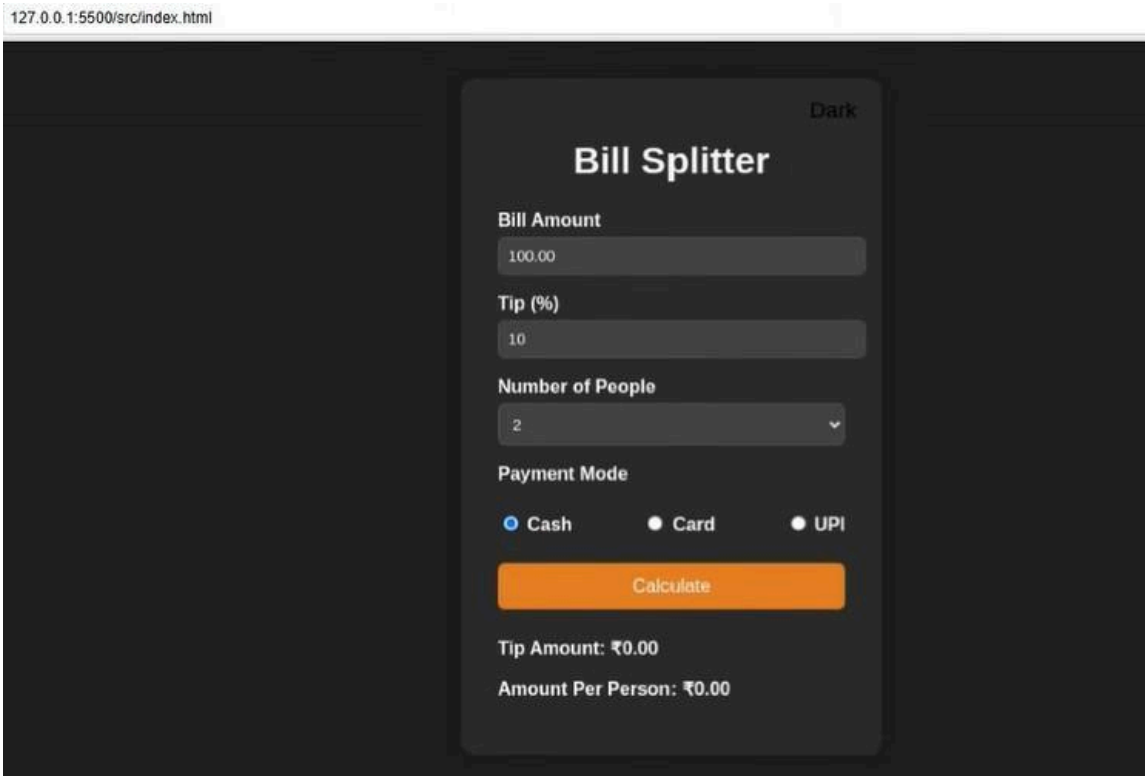
- DOM manipulation using TypeScript.
- Building intuitive UI with semantic HTML structure.
- Implementing layout and styling using CSS techniques.
- Enhancing user experience with interactive features like theme switching.

ScreenShots:

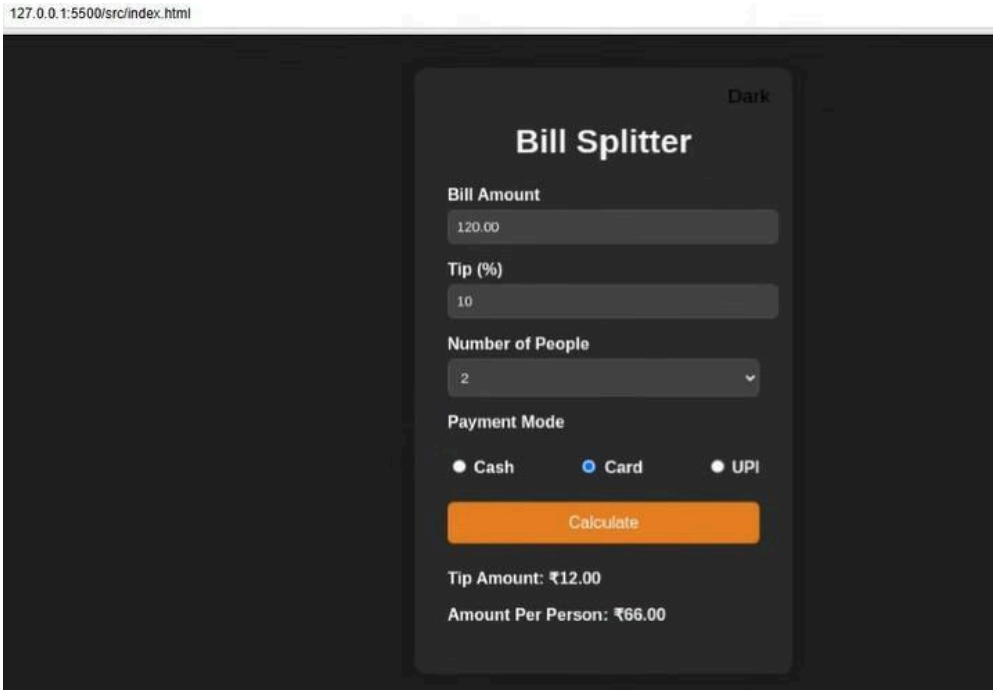
*****Light Mode*****

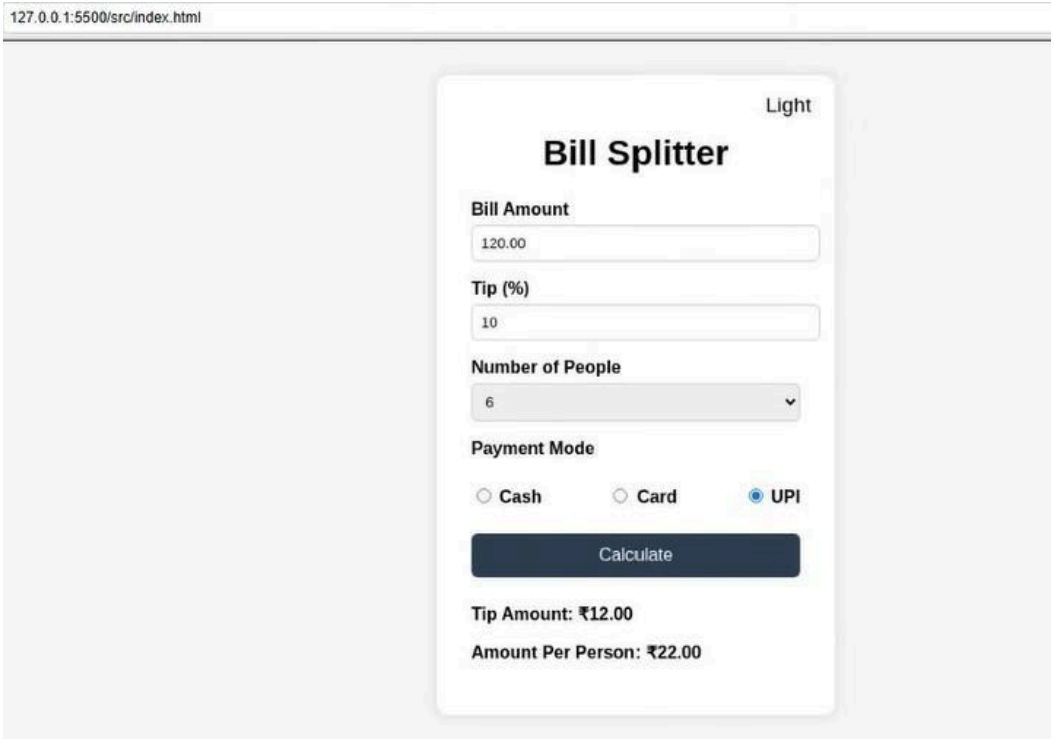


Dark Mode


127.0.0.1:5500/src/index.html
Dark
Bill Splitter
Bill Amount
100.00
Tip (%)
10
Number of People
2
Payment Mode
Cash
Card
UPI
Calculate
Tip Amount: ₹0.00
Amount Per Person: ₹0.00

Calculation





1. index.html

Create the HTML structure for a simple bill-splitting tool using the following specifications:

- The page must include these core HTML elements: `<html>`, `<head>`, `<title>`, `<link>`, `<body>`, `<div>`, `<button>`, `<input>`, `<select>`, `<label>`, `<script>`.
- The `head` should include:
    - A `title` with the text “Bill Splitter”
    - A link to the external CSS file: `style.css`
- Inside the `body`, add a top-level container `div having a class with a name container` to hold all elements.
- Include a **theme toggle button** with the text (“Light” for light mode, “Dark” for dark mode).
- Add three main input fields:
    - Create a label that reads **“Bill Amount”**, followed by an input field with the ID **"amount"** and a default value of **100.00**, allowing users to input the total bill amount.
    - Create a label with the text **“Tip (%)”**, followed by an input field with the ID **"tip"** and a default value of **10**, allowing users to specify the tip percentage to be applied.
    - Create a label that says **“Number of People”**, followed by a dropdown menu with the ID **"people"**, containing options for numbers **1 through 10**, with the option **2** pre-selected — allowing users to choose how many people will split the bill.
- Below these, add a **Payment Mode** section using radio buttons inside a `div` having a `class` name `payment-options.`
- Add a **Calculate** button with id `calculate`
- Below the button, display the results inside a `div` having a `class` name `output.`

    - Add a paragraph displaying the text Tip Amount" show the Tip Amount.
    - Add a paragraph displaying the text "Amount Per Person" to show Amount Per Person.

- 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"

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

---

2. style.css

Add styles for the layout and appearance of the Bill Splitter app:

- Use the universal selector `body` to reset and apply base styles (e.g., font-family, margin, padding).
- Center the `.container` and apply background color, padding, border-radius, and box-shadow.
- Style form elements:
    - `Input` and `Select` should be full-width with padding and border styling.
- Style the **Calculate** button with:
    - Consistent size
    - Background color
    - Rounded corners
    - Hover effect
- Style the **output section** (`.output`) with bold text and spacing.
- Use CSS to handle **theme switching**:
    - Default is light mode

    - Add a `.dark-mode` class to the body for dark mode styling
- Theme toggle button (`#toggle-theme`) should be positioned top-right inside `.container`.

---

3. script.ts

Write the TypeScript logic to calculate and display results, and handle theme switching:

- Define a function `calculateBill()` as:
    - Read input values: bill amount, tip %, and number of people
    - Compute the tip amount and total amount per person
    - Update the result fields (`#tip-amount` and `#per-person`)
- Define a `toggleTheme()` function as:
    - Toggle the `dark-mode` class on the `body`
    - Update the text on the toggle button
- Define a `fillPeopleDropdown()` function.
- Use event listeners:
    - On `#calculate` button to trigger calculation
    - On `#toggle-theme` to switch themes

Detailed HTML Structure Guide for Bill

Splitter App

1. Document Declaration and Root Elements

Start the document by specifying the HTML5 doctype. Then open the HTML structure and set the language attribute to English.

Inside the head section:

- Set the character encoding to UTF-8.
- Add a meta tag for responsive behavior with width set to device-width and initial scale set to 1.0.
- Set the title of the page to “Bill Splitter”.
- Link an external stylesheet pointing to `style.css`.

2. Body Setup

- In the body, apply a class called `light-mode` to allow theme toggling later.
- Create a main wrapper using a division with the class name `container`. All other elements should be placed inside this container.

3. Heading

- At the top of the container, place a level-one heading with the text “Bill Splitter” to act as the title of the application.

4. Theme Toggle Button

Add a button at the top right inside the container to toggle between light and dark themes.

- Assign this button an ID of `toggle-theme`.
- Give it a title attribute with the value "Toggle Theme".
- Set its initial visible text to “Light”.

This button will later change to “Dark” when the dark mode is activated.

5. Bill Amount Input Section

Accepts user input for the bill, create a label that reads “Bill Amount”. Below it:

- Add an input field.
- Set the input’s ID to `amount`.
- Assign a default value of 100.00 to this input field.
- This will allow users to enter the total bill amount.

6. Tip Percentage Input Section

Create another label with the text “Tip (%)”. Below it:

- Add an input field.
- Assign the ID `tip` to this input.
- Set a default value of 10.
- This input lets users define the tip percentage to be applied.

7. Number of People Selection Section

Create a label that says “Number of People”. Below that:

- Add a dropdown menu (select element).
- Assign the ID `people` to this select.
- Populate it with options for numbers 1 through 10.
- Make sure the option with value 2 is pre-selected.

This dropdown allows the user to choose how many people will split the bill.

8. Payment Mode Selection

Below the number of people section, add another label with the text “Payment Mode”.

Then, add a section using a division with the class name `payment-options`. Inside this section:

- Add three radio button inputs for the payment method:
  - ➔ One for “Cash” with value set to "Cash".
  - ➔ One for “Card” with value set to "Card".
  - ➔ One for “UPI” with value set to "UPI".
- Ensure all three radio buttons share the same name attribute, such as `payment`.

- The "Cash" option should be selected by default.

Wrap each radio button inside a label that also displays the option name next to the input.

9. Calculate Button

Below the form fields, insert a button that triggers the calculation.

- Assign this button an ID of `calculate`.
- Set its text to “Calculate”.

This button will perform the tip and per-person calculations when clicked.

10. Output Section

After the Calculate button, create a division with the class name `output`. Inside this section:

- Add a paragraph displaying the text “Tip Amount:” followed by a span.
  - ➔ Set the span’s ID to `tip-amount`.
  - ➔ Initialize the span’s text content to “₹0.00”.
- Add another paragraph for “Amount Per Person:” followed by another span.
  - ➔ Set this span’s ID to `per-person`.
  - ➔ Also initialize it to “₹0.00”.

These spans will be updated dynamically using TypeScript.

11. JavaScript Link

- 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”

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

Detailed CSS Styling Guide for Bill

Splitter App

This guide explains how to style the `style.css` file for the Bill Splitter project. Each section includes the **purpose** of the styles and **how they contribute** to the visual design and usability.

1. Body Styling

Applies the base page styling and centers the app:

- **Font**: `Arial, sans-serif` for clean, readable text.
- **Background color**: `#f5f5f5` (soft light gray for neutral backdrop).
- **Margin**: `0` to reset browser defaults.
- **Padding**: `2rem` around page content for spacing.
- **Display**: `flex` for flexible layout management.
- **Horizontal alignment**: `justify-content: center` to center the container.
- **Transition**: `background-color 0.3s, color 0.3s` for smooth theme changes.

2. App Container (.container)

Styles the white card that holds the app UI:

- **Background color**: `white` for clean visual separation.
- **Padding**: `2rem` inside the box for breathing space.
- **Border-radius**: `12px` for rounded corners.
- **Box-shadow**: `0 0 12px rgba(0,0,0,0.1)` adds soft depth.
- **Max-width**: `300px` keeps layout compact and mobile-friendly.
- **Width**: `100%` for responsiveness.
- **Position**: `relative` to support absolutely positioned elements (like theme toggle).

3. Heading (h1)

Styles the title at the top:

- **Text alignment**: `center`.

- **Bottom margin**: `1.5rem` for spacing from the next section.

4. Labels (label)

Styles form labels:

- **Display**: `block` so labels appear on their own line.
- **Top margin**: `1rem` for separation from prior input.
- **Font-weight**: `bold` for emphasis and clarity.

5. Inputs and Dropdowns (input[type="number"], select)

Creates a consistent input style:

- **Width**: `100%` to fill the container.
- **Padding**: `0.5rem` for a comfortable input area.
- **Margin-top**: `0.3rem` for spacing below labels.
- **Border-radius**: `6px` for modern soft edges.
- **Border**: `1px solid #ccc` for light definition.

6. Calculate Button (#calculate)

Styles the main action button:

- **Width**: `100%` to fill the container.
- **Margin-top**: `1.5rem` for spacing above the button.
- **Padding**: `0.7rem` for easy tap targets.
- **Font-size**: `1rem` for consistent typography.
- **Background color**: `#2c3e50` (dark blue-gray).
- **Text color**: `white` for high contrast.
- **Border**: `none` for flat modern style.
- **Border-radius**: `6px` for rounded shape.
- **Cursor**: `pointer` to indicate clickability.

7. Button Hover (#calculate:hover)

Provides visual feedback:

- **Background color**: `#1a252f` (slightly darker shade).

8. Payment Options Row (.payment-options)

Styles the row of radio buttons:

- **Margin-top**: `0.5rem` for separation from prior field.
- **Display**: `flex` for horizontal layout.
- **Justify-content**: `space-between` to distribute buttons evenly.

9. Output Section (.output)

Displays result values:

- **Margin-top**: `1.5rem` above the result section.
- **Font-weight**: `bold` to highlight the computed output.

10. Theme Toggle Button (#toggle-theme)

Creates the circular icon button at the top right:

- **Position**: `absolute` inside `.container`.
- **Top**: `1rem` and **Right**: `1rem` to place it at the top-right corner.
- **Background**: `transparent` for clean UI.
- **Border**: `none` to remove browser styling.
- **Font-size**: `1.2rem` for emoji/icon visibility.
- **Cursor**: `pointer` to show it's clickable.

11. Dark Mode Base (body.dark-mode)

Applies dark theme to background and text:

- **Background color**: `#1e1e1e` (very dark gray).
- **Text color**: `white` for contrast.

12. Dark Mode Container (body.dark-mode .container)

Darkens the app container:

- **Background color**: `#2b2b2b`.
- **Text color**: `white`.

13. Dark Mode Inputs and Selects

Ensures form fields match the dark theme:

- **Background-color**: `#444` for input fields.
- **Text color**: `white` for visibility.
- **Border-color**: `#555` (medium-dark gray for contrast).

14. Dark Mode Calculate Button

Adds a bright color accent to the main button:

- **Background color**: `#e67e22` (orange).
- **Hover color**: `#cf711b` (deeper orange for interaction feedback).

Detailed TypeScript Logic Guide for Bill

Splitter App

This guide explains the core TypeScript functionality implemented in `script.ts`. Each section describes the role of specific functions and logic used to power the app's interactivity.

1. Main Logic Function: `calculateBill()`

Handles the core calculation logic:

- Gets values from the input fields:
    - Bill amount from the `amount` input.
    - Tip percentage from the `tip` input.
    - Number of people from the `people` dropdown.
- Retrieves output span elements where the results will be shown.
- Converts all values to numbers:
    - Uses `parseFloat` for amount and tip.
    - Uses `parseInt` for number of people.
    - Falls back to default zero if values are invalid.
- Calculates:
    - Tip amount as a percentage of the bill.
    - Total amount (bill + tip).
    - Amount per person (total divided by number of people).
- Updates the inner text of result spans with:

- Tip amount formatted to two decimal places.
- Per-person share formatted similarly.
- Both values prefixed with the Rupee symbol (₹).

2. Theme Toggle Function: `toggleTheme()`

Switches between light and dark UI themes:

- Gets a reference to the toggle button using its ID.
- Adds or removes a `dark-mode` class on the `body` element.
- Updates the label text inside the button depending on current theme:
  - Uses the word “Dark” for dark mode.
  - Uses the word “Light” for light mode.
- Helps users easily toggle visual preferences.

3. Dropdown Setup Function: `fillPeopleDropdown()`

Fills the “Number of People” dropdown dynamically:

- Checks if the people dropdown already contains options to avoid duplicates.
- If empty, creates `<option>` elements for values from 1 to 10.
- Each option’s value and display text is the same (e.g., “3”).
- Sets the default selected value to “2” (a common case for two people splitting a bill).

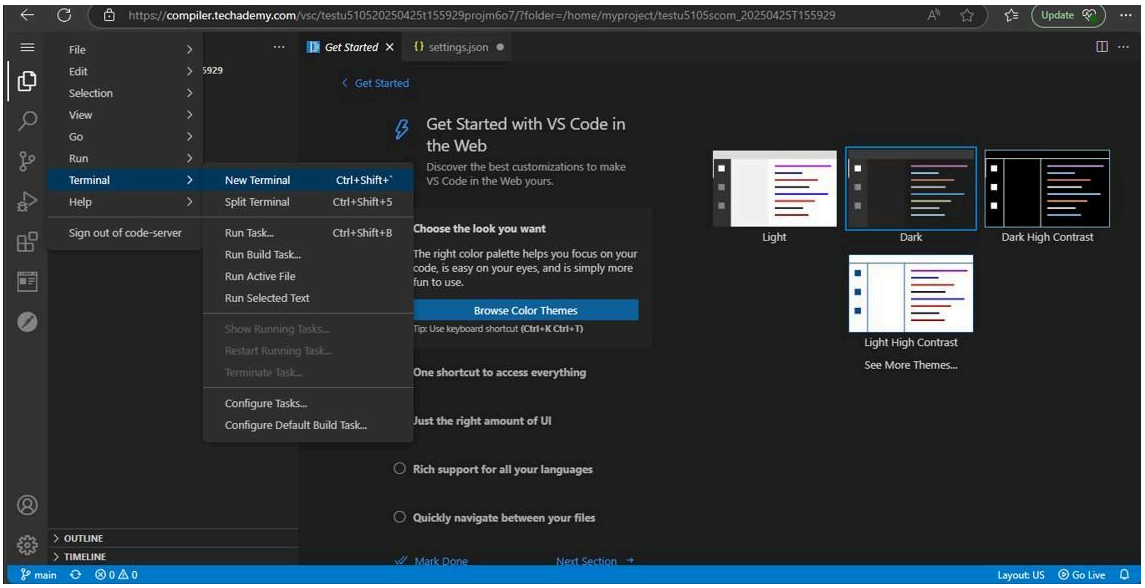
Assessment Guidelines

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

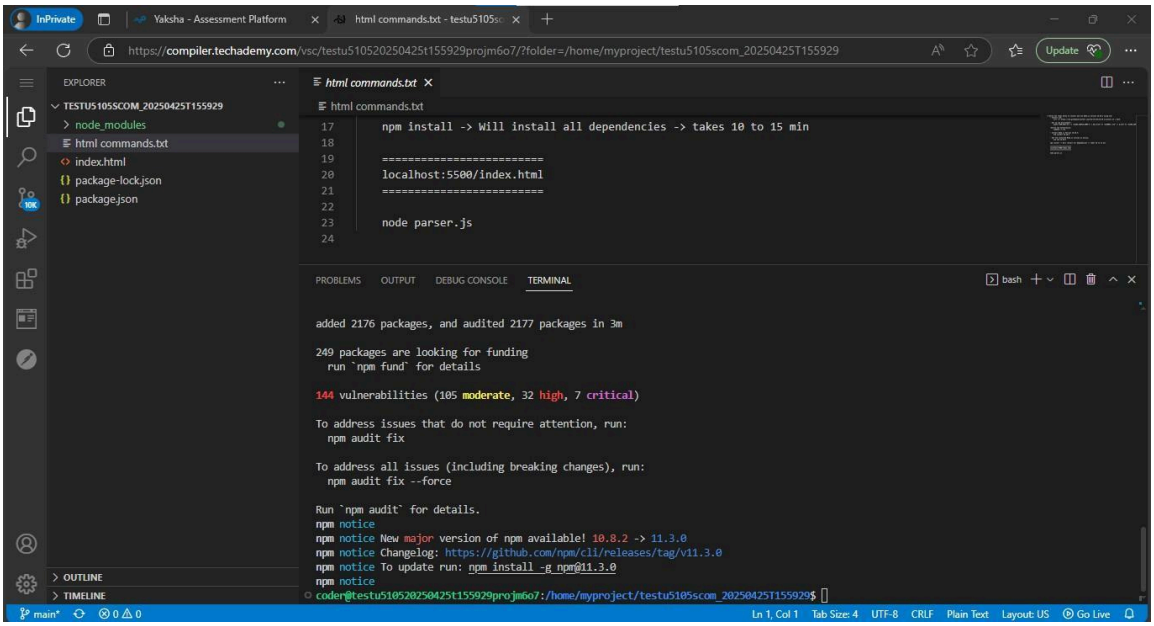
  Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.

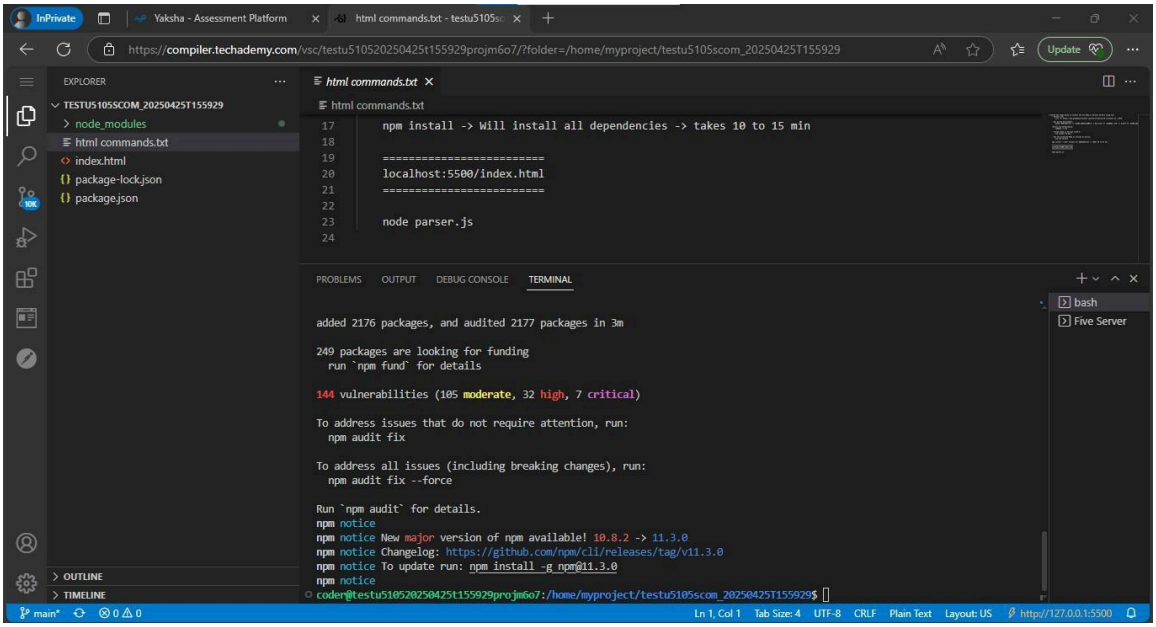
Now in the terminal you need to install all dependencies using the **"npm install"** command.



Step 2:

- Once installation completes, go to the **bottom right corner** of the VS Code screen.
- Click the **"Go Live"** button – This will start a **live server**, The server will run at port `5500` (e.g., `http://localhost:5500/`)

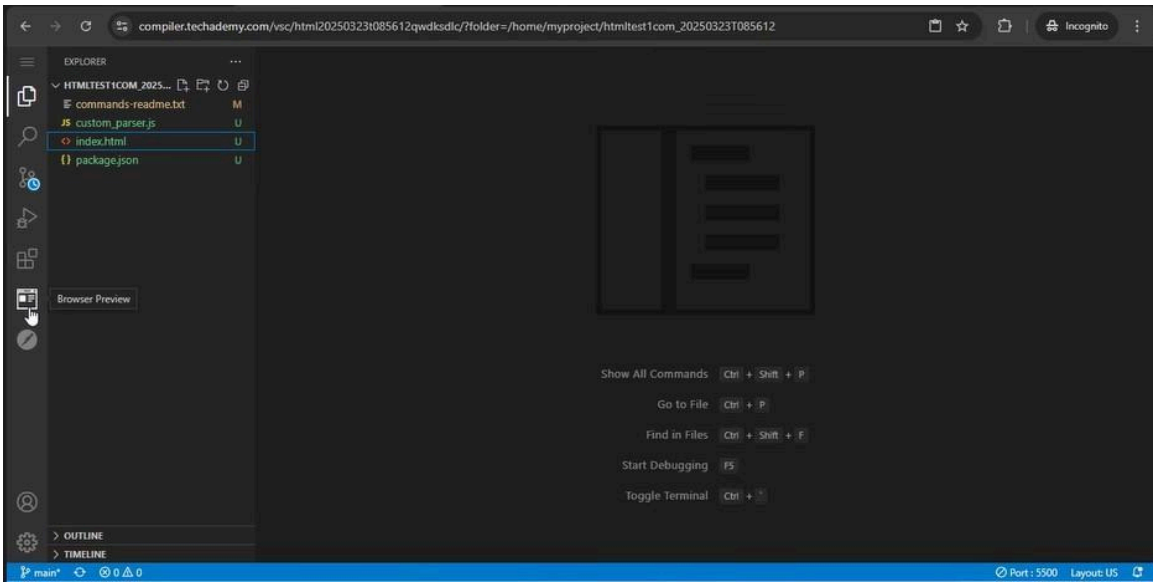




Note: Make sure you run 'npm build run' command from your terminal to generate the JavaScript file from Typescript before running the application, after every change in Typescript file

Step 3: Preview Output in Browser

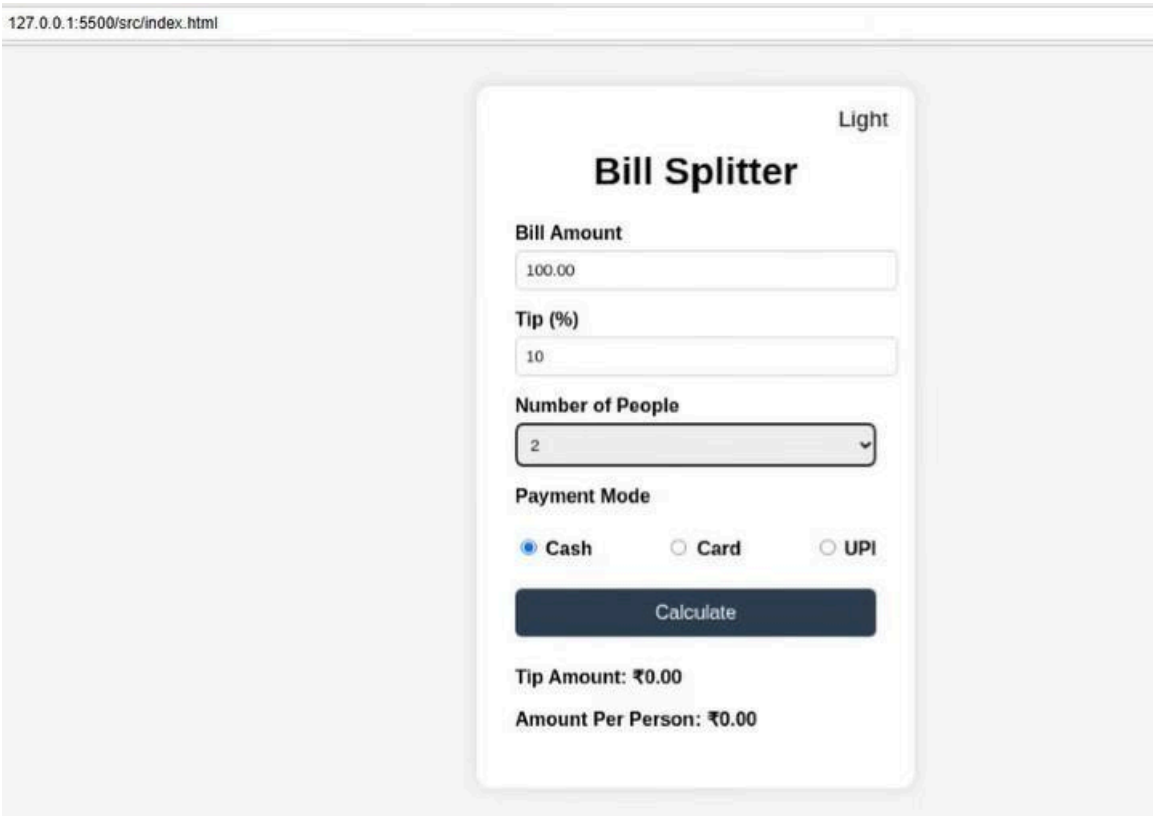
- This is a **web-based application**, so to view it in a browser, use the **internal browser inside the workspace**.
- Click on the **second last icon on the left panel** (the one labeled **"Browser Preview"**). This will open a tab within VS Code where you can **launch and view your application**.
- 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**.



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 a 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 file**
   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.**