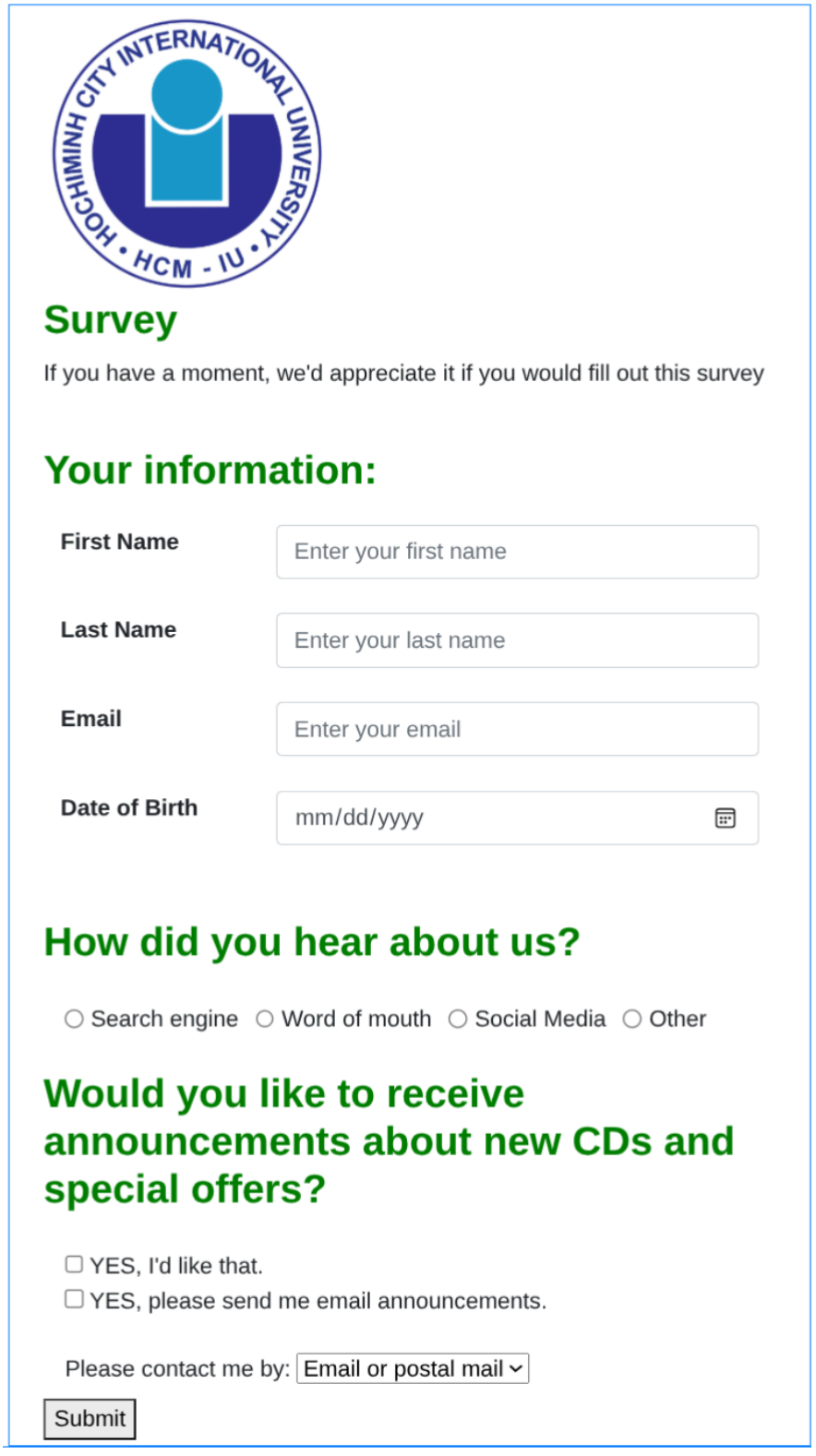
Exercise 2:



1. Functionality

- Input Fields: All basic fields (First Name, Last Name, Email, Date of Birth) are correctly implemented using <input> tags with appropriate type attributes (text, email, date).

- Radio Buttons: The “How did you hear about us?” section uses a shared name attribute, ensuring only one option can be selected at a time.

- Checkboxes: Implemented with <input type="checkbox">, but require name="interests[]" and individual value attributes to correctly submit multiple selections.

- Dropdown Menu: Implemented with the <select> tag and functions properly.

2. HTML Structure

- Overall Structure: The document follows proper HTML5 syntax, including <!DOCTYPE html>, <html>, <head>, and <body>.

- Labels and Attributes: Most fields correctly use <label for="..."> to link labels with inputs, improving accessibility. The checkbox section still lacks proper labels.

- Grouping: Sections can be further improved using <fieldset> and <legend> to semantically group related elements.

3. Visual Design & Layout

- Layout Approach: The form currently uses a combination of <table> for personal information and Flexbox (via Bootstrap) for radio buttons. This approach works but creates visual inconsistency.

- Consistency: Checkboxes, dropdown, and the submit button are not aligned or styled uniformly with the rest of the form. Adding Bootstrap spacing utilities (e.g., .mb-3) will improve overall consistency.

4. Bootstrap Integration

- Library Linking: Bootstrap CSS is correctly linked in the <head>.

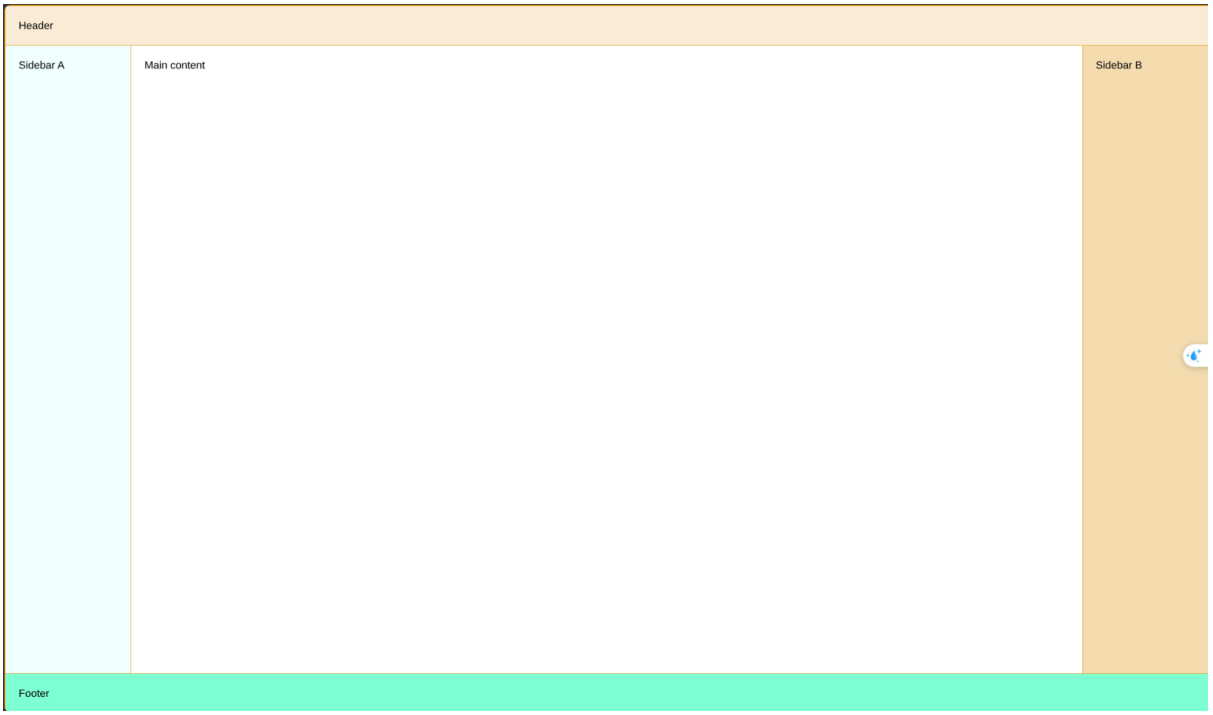
- Class Usage:

  .form-control applied to text inputs.

  .form-check and .form-check-inline applied to radio buttons.

  Bootstrap table classes used for layout.

Exercise 3:



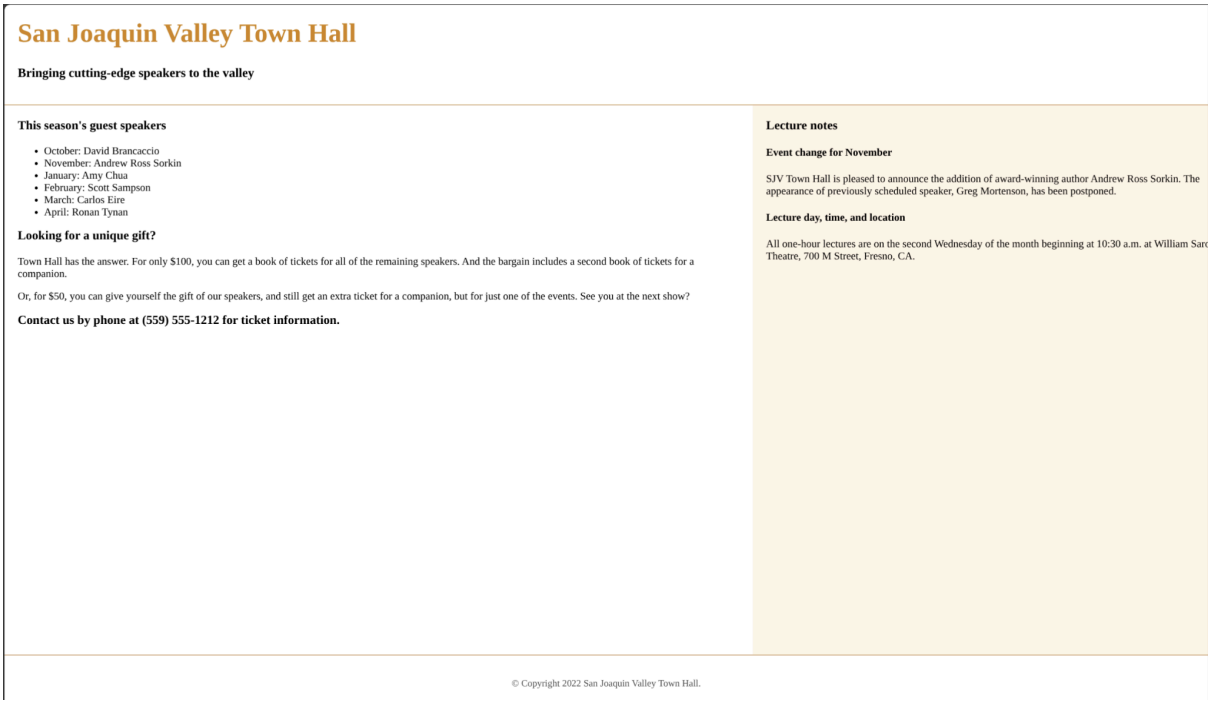
Step 1: Creating the Vertical Layout (Header, Content, Footer)

- Wrap the entire page: Use a parent container such as to enclose all page components.
- Define Flexbox container: Apply display: flex; to the parent container.
- Set vertical direction: Use flex-direction: column; to stack the child elements (<header>, <main>, <footer>) vertically.
- Allow content to grow: Apply flex: 1; (or flex-grow: 1;) to the central content area (<main>) so it expands and fills the remaining vertical space between the header and footer.

Step 2: Creating the 3-Column Horizontal Layout

- Define nested Flexbox container: Set display: flex; on the <main> element. By default, flex-direction is row, which arranges its child columns horizontally.
- Implement fixed-width layout:
- Main content: Apply flex: 1; (or flex-grow: 1;) to make this column flexible and occupy most of the available space.
- Sidebars A and B: Assign fixed widths using either width: 150px; or flex-basis: 150px;. The flex-basis approach is recommended when working with Flexbox layout

Exercise 4:



1. Layout Structure

- The entire page is wrapped inside a parent container: .
- The page contains three main sections:
- <header> – the top section with the page title and subtitle.
- <main> – the central content area of the page.
- <footer> – the bottom section for copyright information.

2. Flexbox Implementation

- Step 1 – Vertical Layout:

    - The .page-container uses display: flex; and flex-direction: column; to arrange the header, main, and footer vertically.
    - min-height: 100vh; makes the layout stretch to fill the full height of the screen.
    - The <main> section uses flex: 1; so it expands to fill the space between the header and footer.

- Step 2 – Horizontal Layout (Columns):

    - The .main-content-area inside <main> is also a Flexbox container (display: flex;) with the default flex-direction: row;.
    - The .main-column takes 60% of the width (flex: 0 0 60%) for the main content.
    - The .sidebar-column takes 40% (flex: 0 0 40%) and has a light background color to visually separate it.