**Course 1: Introduction To Front-End Development**

**Modul 2**

**Topic 1: Getting Started with HTML**

### **Video 1: Summary of HTML Basics and Its Role in Web Development**

1. **HTML as the Frame of a Web Page**:
   * HTML provides the structure of a web page, much like the frame of a building.
   * It consists of tags and elements that define the content and layout.
2. **History of HTML**:
   * Invented by Sir Tim Berners-Lee in 1991 while at CERN.
   * The first version included the initial web browser and web server.
   * Stands for **Hypertext Markup Language**:
     + **Hypertext**: Text containing links to other text.
     + **Markup**: Refers to the use of tags and elements.
3. **HTML Files**:
   * Basic text files with the extension .html.
   * The default page of a website is often named index.html.
4. **HTML Tags and Elements**:
   * **Elements**: Made up of an opening tag, content, and a closing tag.
     + Example: <p> (paragraph) and </p> (closing paragraph).
   * **Self-closing tags**: Tags that don’t require a closing counterpart.
     + Example: <br> (line break).
   * Tags are enclosed in **angle brackets**, e.g., <tag>.
5. **HTML Standards**:
   * Maintained by the **World Wide Web Consortium (W3C)**.
   * The current standard is **HTML5**, which defines the rules and structure for elements and tags.
6. **How Browsers Use HTML**:
   * Browsers read HTML documents to determine what to display.
   * Example: <img> tags display images; <p> tags display paragraphs.
   * By default, the display is basic unless enhanced by **CSS**.
7. **Relationship Between HTML and CSS**:
   * HTML structures the page.
   * CSS (Cascading Style Sheets) styles the page, defining how elements appear visually.

**Key Takeaways:**

* HTML is fundamental for web page creation.
* Tags and elements provide the structure, and standards like those from W3C ensure consistency.
* Understanding HTML is the first step before integrating CSS and JavaScript for dynamic and visually appealing web pages.

### **Video 2: Web Accessibility Summary**

1. **What It Is**:
   * Ensures websites are usable for everyone, including people with disabilities.
   * Covers visual, auditory, cognitive, neurological, physical, and speech disabilities.
2. **Importance**:
   * Inspired by Sir Tim Berners-Lee's vision of universal access.
   * Legally mandated in some regions (e.g., EU Web Accessibility Directive).
3. **Assistive Technologies**:
   * **Screen Readers**: Read content aloud.
   * **Speech Recognition**: Converts speech to text or commands.
   * **Subtitles/Transcripts**: Help with video content.
4. **Best Practices**:
   * Use correct HTML structure (e.g., <p>, <h1>).
   * Avoid improper spacing like multiple line breaks.
   * Plan accessibility early in the project.
5. **Advanced Tools**:
   * Use **ARIA (Accessible Rich Internet Applications)** for complex web apps.

**Key Takeaway**: Accessible websites ensure inclusivity, follow standards, and support assistive technologies.

**Topic 2: Getting Started with CSS**

### **Video 1: Key Takeaways**

1. **HTML**: Provides structure; **CSS**: Defines style and appearance.
2. **CSS Rule Structure**: Selector, declaration block {}, property-value pairs ending with ;.
   * Example:

h1 {

color: gray;

}

1. **Selectors**:
   * **Type Selector**: Targets all elements of a tag (e.g., h1).
   * **ID Selector**: Targets specific elements using #id.
2. **Precedence**:
   * ID selectors override type selectors due to higher specificity.
3. **Linking CSS**: Use <link rel="stylesheet" href="style.css"> in the <head> of the HTML file.
4. **Tool**: Use **VS Code Live Preview** for instant updates.
5. **Practice**: Create rules and test them on your HTML documents.

### **Types of CSS Selectors with Definitions and Examples**

#### **1. Element Selector**

1. **Definition**: This selector is used to apply styles to all elements of a specific type.  
   **Example**:  
   HTML: <p>This is a paragraph.</p>  
   CSS: p { color: blue; }  
   Effect: All <p> elements will have blue text.

#### **2. ID Selector**

1. **Definition**: This selector is used to target a specific element by its unique id attribute, prefixed with #.  
   **Example**:  
   HTML: <p id="intro">This is an introduction.</p>  
   CSS: #intro { font-weight: bold; }  
   Effect: The <p> element with the id="intro" will have bold text.

#### **3. Class Selector**

1. **Definition**: This selector targets elements that share the same class attribute, prefixed with ..  
   **Example**:  
   HTML: <p class="highlight">This is highlighted.</p>  
   CSS: .highlight { background-color: yellow; }  
   Effect: All elements with the highlight class will have a yellow background.

#### **4. Element with Class Selector**

1. **Definition**: This selector targets a specific type of element with a specified class.  
   **Example**:  
   HTML: <p class="note">This is a note.</p>  
   CSS: p.note { color: green; }  
   Effect: Only <p> elements with the note class will have green text.

#### **5. Descendant Selector**

1. **Definition**: This selector targets elements that are nested inside a parent element.  
   **Example**:  
   HTML: <div id="blog"><h1>Blog Title</h1></div>  
   CSS: #blog h1 { color: red; }  
   Effect: The <h1> inside the #blog element will have red text.

#### **6. Child Selector**

1. **Definition**: This selector targets the immediate children of an element, using the > combinator.  
   **Example**:  
   HTML: <div id="blog"><h1>Title</h1><div><h1>Sub Title</h1></div></div>  
   CSS: #blog > h1 { color: orange; }  
   Effect: Only the direct <h1> inside #blog will have orange text, not nested ones.

#### **7. :hover Pseudo-Class**

1. **Definition**: This pseudo-class applies styles when a user hovers over an element with the mouse.  
   **Example**:  
   HTML: <a href="#">Hover over this link</a>  
   CSS: a:hover { text-decoration: underline; color: purple; }  
   Effect: When the link is hovered over, it becomes purple and underlined.

#### **8. Universal Selector**

1. **Definition**: This selector targets all elements on a page.  
   **Example**:  
   HTML: <p>Text</p><div>Box</div>  
   CSS: \* { margin: 0; padding: 0; }  
   Effect: Removes margin and padding from all elements.

#### **9. Attribute Selector**

1. **Definition**: This selector targets elements with a specific attribute.  
   **Example**:  
   HTML: <input type="text" placeholder="Enter text">  
   CSS: input[type="text"] { border: 2px solid blue; }  
   Effect: Only <input> elements with type="text" will have a blue border.

**CSS: Text and Color Overview**

**Colors in CSS**

1. **Usage in Properties**:
   * Colors are applied in properties like color, background-color, etc.
   * Example:

p {

color: blue;

}

1. **Ways to Define Colors**:
   * **RGB Value**:
     + Combines Red, Green, and Blue (0–255 range).
     + Example: rgb(255, 0, 0) for red.
   * **RGBA Value**:
     + Adds an alpha channel for transparency (0.0–1.0).
     + Example: rgba(255, 0, 0, 0.8) (80% opacity).
   * **HSL Value**:
     + Defined by Hue (0°–360°), Saturation (0%–100%), Lightness (0%–100%).
     + Example: hsl(0, 100%, 50%) for red.
   * **Hexadecimal Value**:
     + Uses a # followed by RGB in hexadecimal format.
     + Example: #FF0000 for red.
   * **Predefined Color Names**:
     + Use names like red, blue, green.
     + Example: color: red;.

**Text Properties in CSS**

1. **Text Color**:
   * Property: color.
   * Example:

p {

color: red;

}

1. **Font and Size**:
   * font-family:
     + Specifies font styles with fallbacks.
     + Example:

p {

font-family: "Courier New", monospace;

}

* + font-size:
    - Sets font size.
    - Example:

p {

font-size: 12px;

}

1. **Text Transformation**:
   * Controls capitalization using text-transform.
   * Values: uppercase, lowercase, capitalize, none.
   * Example:

p {

text-transform: uppercase;

}

1. **Text Decoration**:
   * Property: text-decoration.
   * Values: underline, overline, line-through, none.
   * Example:

p {

text-decoration: underline;

}

* + Customization options:
    - text-decoration-line: Specifies type of line (underline, etc.).
    - text-decoration-color: Defines line color.
    - text-decoration-style: Line style (solid, dotted, etc.).
    - text-decoration-thickness: Line thickness.
  + Example:

p {

text-decoration-line: underline;

text-decoration-color: red;

text-decoration-style: solid;

text-decoration-thickness: 5px;

}

### **Video 2: Understanding the CSS Box Model**

The CSS Box Model defines how elements are structured and spaced on a webpage. Every HTML element is treated as a rectangular box, consisting of four main parts:

1. **Content**
   * The core of the element, such as text or images.
   * Size can be customized using properties like:
     + width, height
     + min-width, max-width
     + min-height, max-height
2. **Padding**
   * Extends the content area, creating space inside the element.
   * Calculated as:  
     **Padding Box Width = Content Width + Padding Left + Padding Right**  
     **Padding Box Height = Content Height + Padding Top + Padding Bottom**
   * Defined by:
     + padding-top, padding-bottom, padding-left, padding-right
3. **Border**
   * Surrounds the padding and content, forming an outline.
   * Can be styled with:
     + border-width (e.g., thin, medium, thick)
     + border-style (e.g., solid, dashed)
     + border-color
   * Border size calculations: **Border Box Width = Padding Box Width + Border Left + Border Right**  
     **Border Box Height = Padding Box Height + Border Top + Border Bottom**
4. **Margin**
   * Creates space between the element and neighboring elements.
   * Calculated as:  
     **Margin Box Width = Border Box Width + Margin Left + Margin Right**  
     **Margin Box Height = Border Box Height + Margin Top + Margin Bottom**
   * Set with:
     + margin-top, margin-bottom, margin-left, margin-right

**Pro Tips**

* **Shorthand Properties**: You can use short syntax for padding, border, and margin:

css

Copy code

padding: 10px 20px; /\* Top & Bottom: 10px, Left & Right: 20px \*/

border: 2px solid black; /\* Width: 2px, Style: Solid, Color: Black \*/

margin: 5px; /\* All sides: 5px \*/

* **Visualization Analogy**:
  + **Content**: You.
  + **Padding**: Your clothes.
  + **Border**: Your silhouette or outline.
  + **Margin**: Your personal space.

**Importance**

The box model ensures a clean and structured layout. By manipulating content, padding, borders, and margins, you can create visually appealing and functional designs, ensuring a smooth flow for users navigating your site. Start practicing with the box model to master web layout design!

### **Video 4: Document Flow in Web Browsers**

The default way web browsers position HTML elements on the screen is called **document flow**. Elements are classified into two main categories:

* **Block-level elements**
* **Inline elements**

**Block-level Elements**

* Occupy the full horizontal width of their parent container.
* The vertical height is determined by the content.
* Automatically start on a new line and stack on top of each other.
* Common examples:
  + <div>
  + <p>
  + <form>
  + Headings like <h1> to <h6>

**Example**:

html

Copy code

<div>

<h1>Heading</h1>

<p>Paragraph</p>

</div>

**Inline Elements**

* Only occupy the width and height of their content.
* Do not start on a new line; they flow within surrounding content.
* Common examples:
  + <span>
  + <a> (anchor tag)
  + <img>
  + <input>
  + Formatting tags like <b>, <i>, and <em>

**Example**:

html

Copy code

<p>This is a <span>span element</span> inside a paragraph.</p>

**Changing Display Behavior**

You can switch elements between block and inline using the **CSS display property**.

* To change a block element to inline:

css

Copy code

#example {

display: inline;

}

* To change an inline element to block:

css

Copy code

#example {

display: block;

}

**Demonstration Summary**

1. **Default Behavior**:
   * Block elements start on new lines.
   * Inline elements flow within the same line.
2. **Example in Visual Studio Code**:
   * A <div> with <span> elements (inline) results in unbroken text flow.
   * Changing a <span> to <div> moves the content to a new line due to block-level behavior.
3. **CSS Display Property**:
   * Setting display: inline makes a block element behave like an inline element.
   * Setting display: block makes an inline element behave like a block element.

**Key Takeaways**

* **Block Elements**: Begin on a new line and take up full width.
* **Inline Elements**: Stay in the flow of the content without breaking onto a new line.
* Developers must be comfortable working with both and using the display property to modify layout behavior.

**Key Points on Alignment Basics in CSS**

**Text Alignment**

* Use the text-align property to align text within an HTML element:

css

Copy code

p {

text-align: center;

}

* Alignment options:
  + **Left** (default for LTR languages like English).
  + **Right** (default for RTL languages like Arabic).
  + **Center.**
  + **Justify** (spreads text evenly across the line width).

**HTML Element Alignment**

Aligning HTML elements involves manipulating the **box model** and considering **document flow**.

**Center Alignment**

1. **Block Elements**  
   To center a block element:
   * Define a width.
   * Set margin: auto.

css

Copy code

.child {

width: 50%;

padding: 20px;

border: 4px solid green;

margin: auto;

}

This centers the block-level element horizontally.

1. **Inline Elements**  
   Convert inline elements (like img) to block-level elements:

css

Copy code

.child {

display: block;

width: 50%;

margin: auto;

}

* + Alternatively, use margin-left: auto and margin-right: auto.

**Left / Right Alignment**

1. **Using Float** The float property positions elements relative to the parent content:
   * Example: Align an image to the right:

css

Copy code

img {

float: right;

}

* + - Text wraps around the floated element.

1. **Using Position**
   * The position property provides alternative alignment strategies (covered in more advanced sections).

**Important Notes**

* Center alignment using margin: auto only works for block-level elements.
* Padding can be set using shorthand like padding: 20px;, which applies to all sides equally.
* Fine-tune alignment by adjusting top and bottom margins separately:

css

Copy code

.child {

margin-left: auto;

margin-right: auto;

margin-top: 10px;

margin-bottom: 15px;

}

### **Video 4: Key Points about Bootstrap**

* **Definition:** Library of CSS and JavaScript for quickly building visually appealing websites.
* **Framework Type:** Often referred to as a "front-end framework," "CSS framework," or "CSS library."
* **Components:** Provides reusable components for fast web development.
* **Responsive Grids:** Includes pre-made CSS rules for building responsive layouts adaptable to various devices.
* **Popularity:** Saves development time by offering pre-written CSS classes and components.
* **Ease of Use:** Reduces the need for extensive CSS knowledge; ideal for quick iterations and prototypes.
* **Customizability:** Once understood, Bootstrap's styling can be tweaked for custom designs.
* **Team Collaboration:** Provides a common design system, simplifying teamwork and transitions between projects or companies.
* **Industry Relevance:** A prerequisite skill in many web development roles.
* **Skill Enhancement:** Learning Bootstrap boosts web development skills and productivity.

**Video 5: Using Bootstrap styles**

**Key Points about Bootstrap CSS Class Infixes and Modifiers**

* **Responsive Design:** No need to redesign a website for each device; Bootstrap ensures adaptability across devices, platforms, and screen sizes.
* **Bootstrap CSS Library:** Built using thousands of use cases and structured around reusable CSS classes with infixes and modifiers.
* **Infixes for Breakpoints:** Used to handle responsive breakpoints in the grid system.
  + **Breakpoints and Abbreviations:**
    - Extra Small (<576px): Default, no abbreviation.
    - Small (≥576px): sm
    - Medium (≥768px): md
    - Large (≥992px): lg
    - Extra Large (≥1200px): xl
    - Extra Extra Large (≥1400px): xxl
  + **Mobile-First Approach:** Extra small is the default in Bootstrap.
* **Using Infixes:** Insert the abbreviation (e.g., lg) into the CSS class name for device-specific rules, e.g., col-lg-6 for 6 columns on large screens.
* **Modifiers for Customization:** Used to apply specific styles or colors to elements.
  + Example: Alert modifiers like alert-primary (blue) or alert-danger (red) change the appearance of alerts.
  + Modifier Options: primary, secondary, success, info, warning, danger, light, dark.
* **Predefined Components:** Bootstrap provides prebuilt UI styles and elements such as alerts for user notifications (e.g., warnings, errors, confirmations).
* **Alerts Example:**
  + To display an alert with a blue background: alert alert-primary.
  + To change the alert to red for an error: Replace primary with danger.
* **Efficiency:** Use of infixes and modifiers reduces development time and enables rapid UI styling and layout adjustments.

**Video 5: Bootstrap Grid**

**Key Points on Bootstrap Grid System**

* **Responsive Design:**
  + Bootstrap enables responsive design using a **grid system** and **responsive breakpoints** to adapt layouts across devices.
* **12-Column Grid System:**
  + Bootstrap uses a **12-column grid** that can be **fluid** (responsive) or **fixed** (static width).
  + Content is organized into **containers**, **rows**, and **columns**.
* **Containers:**
  + The **container** is the root element of the grid, responsible for padding and aligning content.
  + Container width adapts based on responsive breakpoints.
* **Rows and Columns:**
  + Rows house columns and structure the content.
  + Columns define the width of the content and are adjustable with suffixes (e.g., col-4 for 4 spaces and col-8 for 8 spaces).
* **Breakpoint-Specific Layouts:**
  + Use **breakpoints** to configure layouts for different devices:
    - Example: col-12 for mobile (stacked layout).
    - Example: col-lg-6 for desktop (side-by-side layout).
  + Columns automatically wrap if the total column spaces exceed 12.
* **Simulating Device Layouts:**
  + Use browser developer tools (e.g., F12 key) and enable device mode to test layouts on various screen sizes like mobile and tablets.
* **Efficiency:**
  + Bootstrap’s grid system eliminates the need to develop separate layouts for each device, saving significant development time.
* **Real Example:**
  + Columns can be adjusted ally:
    - For mobile: Stack content by setting columns to span 12 spaces (col-12).
    - For desktop: Arrange content side by side using col-lg-6.
* **Powerful and Flexible:**
  + Suitable for most development needs due to its adaptability and pre-defined CSS rules.

**Topic 3: Introduction to React**

**Video 1: Static and Dynamic Content**

 **Static vs. Dynamic Content:**

* **Static Content:** Files like images or videos transferred as they are stored on the web server.
* **Dynamic Content:** Generated when an HTTP request is made, based on user input or real-time factors like the current date.

 **Roles of Web Server and Application Server:**

* **Web Server:**
  + Sends content (static or dynamic) to the user's browser.
  + Handles caching to improve performance.
* **Application Server:**
  + Generates dynamic content based on logic, user input, or database queries.
  + Performs complex tasks like checking permissions, running application logic, and interacting with databases.

 **Static Content Example:**

* Watching a video involves a simple request to the web server, which sends the video file to the browser.

 **Dynamic Content Example:**

* Logging into a course:
  + Web server sends a request to the application server.
  + Application server checks enrollment, retrieves relevant data, and specifies personalized content for the user.

 **Caching for Performance Improvement:**

* **Process:**
  + On the first request, dynamic content is fetched from the application server and stored in the cache.
  + Subsequent requests are served directly from the cache, reducing load on the application server.
* **Cache Updates:** Content in the cache is periodically refreshed to ensure it remains up-to-date.

 **Performance Challenges:**

* Dynamic content generation is slower and more resource-intensive than serving static content.
* Application servers have limited capacity for processing requests per second, making caching essential.

 **Specialized Application Servers:**

* Tailored for specific purposes, such as music streaming or blogging.
* Developers can create custom application servers for their needs.

 **Takeaway:**

* Static content is fixed, while dynamic content is personalized and generated on-demand.
* Web servers and application servers work together, with caching playing a crucial role in optimizing performance.

**Video 2: Single Page Applications**

 **Traditional Websites:**

* Traditional websites are multi-page applications (MPAs) where each user request triggers the server to send a full new HTML page.
* This can be resource-intensive, leading to slower user experiences, especially with complex sites or poor internet connections.

 **Single-Page Applications (SPAs):**

* SPAs send only one HTML page from the server to the browser, which dynamically updates content as users interact.
* The page doesn’t need to reload for every action, making the experience faster and more responsive.

 **How SPAs Work:**

* **Bundling**: All necessary HTML, CSS, and JavaScript are loaded at once, which can lead to slower load times for large applications.
* **Lazy Loading (Code Splitting)**: Only the minimum resources are loaded initially; additional resources are loaded as needed, which is more efficient for larger applications.

 **Example (Movie Review Site):**

* **Traditional Website**: When a button is clicked, a post request is made to the server, which returns a new web page with the updated content.
* **SPA**: When the button is clicked, the server returns a JSON object, and the SPA updates the current page without reloading it.

 **Multiple Page Content:**

* In traditional websites, navigating between different pages (like news and profile) reloads the entire page.
* In SPAs, pages are divided into templates (views), and only the data (in JSON format) is sent from the server. The SPA dynamically updates the content by inserting the new data into the existing page.

 **Choosing Between Traditional Websites and SPAs:**

* SPAs are more efficient for applications requiring dynamic updates, as they only load necessary data, reducing page reloads and improving speed.
* For complex applications, consider using **lazy loading** to load resources dynamically, while for simpler apps, **bundling** might be sufficient.

 **Performance Considerations:**

* SPAs can offer faster, more engaging experiences, but they require careful resource management (bundling vs. lazy loading).
* The performance depends on the complexity and bandwidth needs of the application.

**Video 3 : What is React?**

 **React Overview**:

* Open-source JavaScript library for building user interfaces for web and mobile apps.
* Focuses on components, reducing code and improving maintainability and testing.

 **React Components**:

* Components are reusable UI pieces (e.g., profile picture, music player).
* Helps developers work on parts of the app in isolation and reuse components.

 **Efficiency with Reusable Components**:

* Reduces repetitive coding by reusing components in different parts of the app.
* Pre-made component libraries are available for tasks like video players or maps.

 **React Community and Growth**:

* Large, growing community with continuous updates and new features.
* Annual conferences foster sharing and development.

 **Purpose of React**:

* React simplifies front-end development by using reusable components, making app development more efficient.

**Video 3: how React Works?**

* **React and the DOM**:
  + React components correspond to HTML elements displayed on the webpage.
  + Updating the browser DOM is time-intensive and can slow down performance.
* **Virtual DOM**:
  + React creates its own in-memory representation of the browser DOM, called the *virtual DOM*.
  + The virtual DOM ensures faster updates and responsiveness by minimizing browser DOM changes.
* **Reconciliation Process**:
  + React compares the virtual DOM with its previous version to detect changes.
  + Only changed elements are updated in the browser DOM, improving performance.
* **Efficiency**:
  + By using the virtual DOM, React avoids unnecessary updates, ensuring applications remain fast and responsive.

**Video 4: The Virtual Dom**

 **Virtual DOM**:

* React creates a virtual DOM, an in-memory representation of the browser DOM.
* React compares the virtual DOM with the browser DOM during updates, only making changes when necessary to improve performance.

 **Reconciliation Process**:

1. Virtual DOM is updated.
2. Virtual DOM is compared to the previous version to identify changes.
3. Only changed elements are updated in the browser DOM.
4. The webpage updates to match the browser DOM.

 **React Fiber Architecture**:

* React Fiber enables incremental rendering by spreading updates over time instead of updating all at once.
* It optimizes updates based on priority:
  + Visible elements are updated first.
  + Non-visible elements are updated later.
* This approach significantly improves performance and responsiveness.

 **Practical Scenario**:

* Updates to non-visible parts of the page (e.g., the top of a long webpage) are deprioritized, focusing instead on visible changes.

 **Developer Tools**:

* Tools like the React Developer Tools browser plugin help investigate how React processes the webpage and resolve issues.

**Video 5: React Components**

**React Component Hierarchy Basics**

* Every React application has at least one **root component**, often called the **App component**.
* Child components are added to the root component, forming a **tree structure** of components.

**Component Hierarchy Example 1: Shopping List**

* **App Component**: Root component.
  + **NewItemBar Component**: Allows users to add items to the shopping list.
  + **ShoppingList Component**: Contains multiple **ShoppingItem Components**, one for each item.
* **Reusability**:
  + Even if items differ (e.g., "chicken" or "noodles"), the same **ShoppingItem Component** can be reused to display them.
  + When an item is checked off, the corresponding **ShoppingItem Component** is removed.

**Component Hierarchy Example 2: Blog Website**

* **App Component**: Represents the entire webpage.
  + **Navbar Component**: Contains the blog's title, navigation links, and a **Search Component**.
  + **Page Component**: Contains blog content:
    - **MainFeature Component**: Displays a featured blog post.
    - **SmallFeature Component** (used twice): Displays smaller blog summaries and includes a **Thumbnail Component** for images.
* **Code Reusability**:
  + The **SmallFeature Component** is reused with different properties for each instance.

**Advantages of React Components**

* **Reusability**: Components can be reused multiple times, saving development time.
* **Modularity**: Breaking down a website into smaller components makes it easier to plan and manage.
* **Efficiency**: Once you master structuring components, building applications becomes faster and more efficient.

**Key Takeaway**

Planning an application as a series of components may feel tricky initially, but with practice, it becomes intuitive and highlights React’s strengths in building user interfaces.

**Alternatives to React & Common Libraries Used with React**

**1. Lodash**

* A utility library that provides common logic to save development time.
* Examples: Sorting lists, rounding numbers, etc.

**2. Luxon**

* Helps manipulate and display dates and times.
* Supports formatting dates for different regions (e.g., US: MM/DD/YYYY, Europe: DD/MM/YYYY).

**3. Redux**

* Manages the application state.
* Useful for tracking data like items in a shopping cart.
* Offers advanced features like undo and redo.

**4. Axios**

* Simplifies sending HTTP requests and processing responses.
* Features include canceling requests and transforming response data before use.

**5. Jest**

* A library for writing automated tests.
* Works with many libraries and frameworks.
* Includes reporting tools to check how much code is covered by tests.

**Conclusion**

* Each library focuses on solving specific problems in application development.
* Official websites offer setup guides, tutorials, and documentation for further exploration.