1 Introduction to Tailwind CSS & Utility first Workflow Tailwind Tutorial

Utility first framework :

In web development, a utility-first framework provides a set of low-level CSS classes that you can combine to achieve specific design goals. These classes focus on a single styling aspect, like adding padding or setting a background color.

Here's a breakdown of the key concepts:

- **Utility Classes:** These are the building blocks of a utility-first framework. They are small, specific CSS classes that target a single styling property, for example, a class named `.text-red` might set the text color to red.
- **Custom Design:** Unlike some other frameworks that come with pre-built components, utility-first frameworks give you the freedom to create entirely custom designs by combining these utility classes directly in your HTML.
- **Benefits:**
  - **Rapid Development:** Since you don't need to write a lot of CSS yourself, utility-first frameworks can speed up the development process.
  - **Flexibility:** You have fine-grained control over the styles of your elements.
  - **Maintainability:** Utility classes are often self-documenting due to their clear names, making your code easier to understand and update.
- **Drawbacks:**
  - **Verbosity:** Your HTML code can become cluttered with many class names.
  - **Complexity:** For intricate designs, managing numerous classes can become cumbersome.

A popular example of a utility-first framework is Tailwind CSS https://tailwindcss.com/docs/installation. It offers a vast collection of utility classes that you can use to style your web pages.

Vanilla JavaScript

Vanilla JavaScript, also referred to as Vanilla JS, is simply JavaScript in its purest form, without the use of any external libraries or frameworks. It leverages the core functionality that's built directly into web browsers.

Here's a breakdown of the concept:

- **Pure JavaScript:** Vanilla JS relies solely on the features and functionalities that the JavaScript language itself provides.
- **No External Libraries/Frameworks:** It doesn't incorporate any pre-written code or structures from external sources like jQuery, React, or Angular.
- **Advantages:**
  - **Lightweight:** Vanilla JS is known for its speed and efficiency as it avoids the additional load of external libraries.
  - **Flexibility:** You have complete control over the code, allowing for tailored solutions.
  - **Learning Foundation:** Understanding vanilla JS is essential for grasping more complex JavaScript frameworks.
- **Disadvantages:**
  - **Repetitive Coding:** You might end up writing more code to achieve functionalities that libraries or frameworks offer readily.
  - **Complexity for Large Projects:** For extensive web applications, managing vanilla JS can become cumbersome.

While frameworks can streamline development, vanilla JS offers a strong foundation for understanding core JavaScript concepts.

Utility class :

A utility class in CSS is a pre-defined class that offers specific styling options without relying on complex selectors or custom class names. They act like building blocks for your web page's design.

Here's a closer look at utility classes:

Characteristics:

- **Single-Purpose:** Each utility class typically targets a single CSS property, like `padding`, `margin`, `font-size`, or `color`.
- **Descriptive Naming:** Utility classes often have clear and concise names that reflect their purpose. For example, a class named `.text-red` would apply a red color to the text.
- **Reusable:** The same utility class can be applied to various HTML elements to achieve consistent styling throughout your website.

Benefits of Using Utility Classes:

- **Rapid Development:** Quickly style elements using pre-defined classes instead of writing custom CSS for each element.

- **Maintainable Code:** Utility classes promote cleaner and more readable CSS code. Descriptive names make it easy to understand what styles are applied.
- **Consistent Design:** Using the same utility classes across components ensures a visually cohesive website.
- **Responsive Design:** Utility classes often come in variations for different screen sizes, making it easier to create responsive layouts.

Example:

Here's an example of how you might use utility classes to style a button:

HTML

```html
<button class="bg-blue-500 text-white font-bold py-2 px-4 rounded">Click
Me</button>
```

content_copy

This code applies several utility classes to the button:

- `.bg-blue-500`: Sets the background color to blue (shade defined by the class).
- `.text-white`: Sets the text color to white.
- `.font-bold`: Makes the text bold.
- `.py-2`: Applies padding to the top and bottom of the button (size defined by the class).
- `.px-4`: Applies padding to the left and right of the button (size defined by the class).
- `.rounded`: Adds rounded corners to the button.

Comparison to Traditional CSS:

Traditionally, you might style a button using a custom class like `.my-button`. This class would then be defined in your CSS file with all the necessary styles. While this approach works, it can lead to repetitive code and less readable CSS.

Utility-First Frameworks:

Tailwind CSS is a popular example of a utility-first framework. It provides a vast collection of utility classes that cover a wide range of styling options. You can pick and combine these classes to achieve the desired look for your web pages.

In conclusion, utility classes offer a powerful and efficient way to style your web pages. They promote rapid development, maintainable code, and consistent design.

Plugin:

A plugin, also referred to as an add-on, add-in, or extension, is a software component that adds specific features or functionalities to an existing program. They essentially act as modular additions that enhance the capabilities of the host program without requiring modifications to its core code.

Here's a breakdown of how plugins work:

- **Designed for Expansion:** Plugins are created by developers to extend the functionality of existing software. This allows users to customize their experience and perform tasks that might not be possible with the base program alone.
- **Integration through Interfaces:** Plugins work by integrating with the host program through defined interfaces or APIs (Application Programming Interfaces). These interfaces act as communication channels, allowing the plugin to interact with the program's functionalities.
- **User Interface Enhancements:** When you install a plugin, it may add new options to the program's user interface. This could include new menu items, buttons, or configuration panels that allow you to control the plugin's behavior.

Examples of Plugins:

- **Web Browsers:** Web browsers like Chrome, Firefox, and Safari often use plugins to enable features like playing flash videos, managing passwords, or blocking advertisements.
- **Image Editing Software:** Photo editing software like Photoshop can be extended with plugins that offer specialized effects, filters, or tools for specific tasks.
- **Content Management Systems (CMS):** Popular CMS platforms like WordPress allow users to install plugins that add functionalities like contact forms, e-commerce capabilities, or social media integration.

Benefits of Plugins:

- **Customization:** Plugins allow users to personalize their software experience and tailor it to their specific needs.
- **Extensibility:** They offer a way to expand the capabilities of existing software without requiring a complete overhaul of the program itself.
- **Innovation:** The plugin ecosystem fosters innovation as developers can create specialized tools to address various user requirements.

Things to Consider with Plugins:

- **Compatibility:** Ensure the plugin is compatible with your specific version of the host program.
- **Security:** Only install plugins from trusted sources to avoid malware or security vulnerabilities.

- **Performance:** Some plugins can consume resources and slow down your program. Be selective about which plugins you install.

In conclusion, plugins are valuable tools that enhance the functionality and user experience of various software programs. They provide a flexible and modular way to customize your software environment and perform tasks that might not be possible otherwise.

Postcss :

PostCSS is a tool that lets you transform CSS with the help of JavaScript plugins. It essentially acts as a bridge between modern CSS features and browser compatibility.

Here's a breakdown of what PostCSS does:

- **Transforms CSS:** It takes your regular CSS code and converts it into a format that plugins can understand. This format is called an Abstract Syntax Tree (AST).
- **JavaScript Plugins:** A vast ecosystem of plugins exists for PostCSS, each offering specific functionalities. You can choose plugins to add vendor prefixes for wider browser compatibility, write future-proof CSS with features not yet supported by all browsers, optimize stylesheets for size, and more.
- **Not a Preprocessor:** Unlike Sass or Less, which introduce their own syntax for writing CSS, PostCSS works with standard CSS. It leverages plugins to extend its capabilities.

Benefits of Using PostCSS:

- **Modern CSS Today:** Use cutting-edge CSS features while ensuring compatibility with older browsers through plugins.
- **Streamlined Workflow:** Automate repetitive tasks like adding vendor prefixes, improving developer experience.
- **Flexibility:** Choose only the plugins you need, creating a customized CSS workflow.
- **Wide Adoption:** Popular among developers and used by major websites for efficient CSS management.

In essence, PostCSS empowers you to write modern, maintainable CSS while ensuring it works seamlessly across different browsers.

The key takeaway here is that PostCSS helps you use new and exciting features in CSS while still supporting older web browsers that might not understand them yet. It acts like a translator between the two worlds.

Here's a more detailed explanation:

1. **Modern CSS Features:** CSS is constantly evolving, with new features being added all the time. These features offer more powerful and efficient ways to style your web pages.

2. **Browser Compatibility Issue:** The problem is that not all browsers support all the latest CSS features immediately. Older browsers might not understand the code and render your web page incorrectly.
3. **PostCSS as a Bridge:** This is where PostCSS comes in. It takes your modern CSS code and uses plugins to modify it for browsers that don't understand the new features. Here's how it achieves this:
    - **Vendor Prefixes:** Many modern CSS features have vendor prefixes attached in the initial stages. For example, `flexbox` might have different prefixes for different browsers like `-webkit-flexbox` or `-moz-flexbox`. PostCSS plugins can automatically add these prefixes to your code, ensuring compatibility with a wider range of browsers.
    - **Polyfills (Optional):** In some cases, plugins might even inject JavaScript code (polyfills) to mimic the functionality of the new feature for unsupported browsers. This is a more complex approach but can be necessary for critical features.
    - **Fallbacks:** Plugins might also help you define fallback styles that will be used by browsers that don't understand the new feature at all. This ensures some level of functionality even in older browsers.
      By using PostCSS and its plugins, you can write modern CSS code that takes advantage of the latest features without worrying about compatibility issues. PostCSS essentially translates your code for older browsers, making your website look and function as intended across a wider audience.

---

what is the difference between postcss and tailwind css :

PostCSS and Tailwind CSS are both tools used in CSS development, but they serve different purposes:

PostCSS

- **Function:** PostCSS is a **CSS processing tool**. It acts as a middleman between your written CSS and the final output that goes to the browser.
- **Functionality:** PostCSS itself doesn't add styles, but it lets you use **JavaScript plugins** to achieve various tasks. These plugins can:
    - **Transform CSS:** Convert modern CSS features to a format compatible with older browsers (using vendor prefixes).
    - **Optimize Stylesheets:** Reduce file size by removing unused styles or minifying code.
    - **Add Features:** Implement functionalities not yet built-in to CSS (like custom functions or nesting).
- **Benefits:**
    - **Flexibility:** Use only the plugins you need for a customized workflow.

    - **Modern CSS:** Write future-proof CSS with features not supported by all browsers yet (handled by plugins).
    - **Streamlined Workflow:** Automate tasks like adding vendor prefixes.

Tailwind CSS

- **Function:** Tailwind CSS is a **utility-first CSS framework**. It provides a large collection of pre-defined utility classes that you can use to style your web pages.
- **Functionality:**
    - Offers a vast collection of CSS classes representing styles like margins, paddings, colors, fonts, and more.
    - Classes are highly specific and follow a consistent naming convention.
    - You combine these classes in your HTML to achieve the desired look.
- **Benefits:**
    - **Rapid Development:** Quickly build UIs with pre-made classes.
    - **Responsive Design:** Easily create responsive layouts with utility classes for different screen sizes.
    - **Maintainable Code:** Classes are clear and descriptive, promoting clean and readable CSS.

Here's an analogy:

- **PostCSS:** Imagine PostCSS as a powerful toolbox containing various tools (plugins) for working with CSS. You choose the tools you need for a specific task.
- **Tailwind CSS:** Tailwind CSS is like a pre-built furniture set with all the pieces (classes) needed to assemble different styles. You simply pick and combine the pieces to create your desired look.

In short:

- Use PostCSS for **processing and extending** your existing CSS code.
- Use Tailwind CSS for **rapidly building UIs** with pre-defined utility classes.
- You can even **combine PostCSS with Tailwind CSS** for a powerful workflow. Tailwind itself is built using PostCSS plugins

---

what does Install Tailwind CSS using PostCSS mean :

Installing Tailwind CSS using PostCSS signifies setting up Tailwind to work alongside PostCSS in your project. Here's a breakdown of what it involves:

1. PostCSS as the Foundation:

- PostCSS acts as the base tool. It allows you to leverage plugins to manipulate and enhance your CSS code.

2. Tailwind CSS as a Plugin:

- Tailwind CSS itself is a PostCSS plugin. It doesn't directly modify your CSS but provides a collection of utility classes.

3. Installation Process:

- You'll use package managers like npm or yarn to install both PostCSS and Tailwind CSS along with its dependencies.
- During installation, Tailwind creates configuration files (`tailwind.config.js`) to define how it generates utility classes based on your preferences.

4. Integration with Your CSS:

- You'll configure PostCSS to understand Tailwind as a plugin.
- In your main CSS file, you'll import directives provided by Tailwind to include the utility classes you want to use.

Benefits of this approach:

- **Flexibility:** PostCSS offers a modular approach. You can use other plugins alongside Tailwind for tasks like adding vendor prefixes or minifying CSS.
- **Modern Workflow:** Tailwind leverages PostCSS for processing, ensuring compatibility with your chosen plugins.

In essence, installing Tailwind CSS using PostCSS integrates Tailwind's utility classes seamlessly into your project's workflow, allowing you to style your web pages efficiently.

NPM :

npm is the world's largest software registry for JavaScript packages, making development faster and easier.

Here are some examples of popular JavaScript packages and their uses:

- **React (library):** A powerful library for building dynamic user interfaces (UI) for web applications. It allows you to create reusable UI components and manage their state effectively.
- **Lodash (utility library):** Often referred to as a "Swiss Army Knife" for JavaScript, Lodash offers a comprehensive collection of utility functions for common tasks like

array manipulation, object manipulation, string manipulation, and more. It helps write concise and efficient code.

- **Express.js (framework):** A popular framework for building web servers and APIs using Node.js. It provides a streamlined way to handle incoming requests, route them to appropriate handlers, and send responses.
- **Axios (promise-based HTTP client):** Simplifies making HTTP requests (like fetching data from APIs) in JavaScript applications. It offers a clean and intuitive API for sending GET, POST, PUT, and DELETE requests, and handles responses with Promises for asynchronous handling.
- **Moment.js (library):** A library specifically designed for working with dates and times in JavaScript. It provides functionalities for parsing, manipulating, formatting, and validating dates and times, making it easier to deal with date-related tasks in your applications.
- **Bootstrap (framework):** A popular front-end framework that provides pre-built CSS styles and JavaScript components for building responsive websites and web applications. It offers a foundation for building user interfaces with consistent styling and layout options.
- **Vue.js (library):** Another powerful library for building user interfaces. Similar to React, it allows you to create reusable UI components and manage their state. It's known for its ease of learning and adoption.

These are just a few examples, and the npm registry holds a vast collection of packages catering to various development needs.

packages vs libraries vs frameworks

Here's a breakdown of the key differences between packages, libraries, and frameworks:

Level of Abstraction:

- **Packages:** The most basic building block. They often contain a small collection of functions or utilities that perform specific tasks. Think of them as individual tools in a toolbox.
- **Libraries:** A collection of related functionalities grouped together to address a broader need. They offer a wider range of tools for a specific domain or purpose. Imagine a pre-configured toolset for carpentry.
- **Frameworks:** Provide a more comprehensive structure and enforce a particular way of developing applications. They offer a structured environment with pre-defined components and patterns, dictating how you build your application. Think of them as a pre-built house frame that you can customize.

Flexibility vs. Control:

- **Packages:** Offer the most flexibility. You can choose individual functionalities from a package and integrate them into your existing code.
- **Libraries:** Provide a balance between flexibility and control. You can leverage pre-built components and functions while still having some control over how they are used.
- **Frameworks:** Enforce a specific way of doing things, offering less flexibility but often faster development for projects that fit the framework's structure.

Learning Curve:

- **Packages:** Generally have a lower learning curve as they deal with specific tasks.
- **Libraries:** May have a slight learning curve depending on the complexity of the library.
- **Frameworks:** Typically have a steeper learning curve as you need to understand the framework's architecture and design patterns.

Here's an analogy:

Imagine you're building a house:

- **Packages:** Individual bricks, nails, or hammers.
- **Libraries:** A pre-built toolbox with carpentry tools like saws, drills, and screwdrivers.
- **Framework:** A pre-fabricated house frame with walls, floors, and a roof. You can customize the interior but the basic structure is vorgegeben (German for "given").

Choosing the Right Tool:

- **Packages:** Use them for specific tasks that don't require a full-blown library or framework.
- **Libraries:** A good choice when you need a collection of related functionalities for a particular domain.
- **Frameworks:** Ideal for projects that benefit from a structured approach and rapid development if the project aligns with the framework's structure. You can even combine libraries and packages within a framework for more granular control.

In conclusion, packages, libraries, and frameworks are all essential parts of the JavaScript developer's toolkit. Understanding their differences helps you choose the right tool for the job, ensuring efficient and maintainable development.

NPX :

npx, which stands for **Node Package Execute**, is a tool included with npm (Node Package Manager) that allows you to execute commands from packages in the npm registry without having to install them globally. Here's a breakdown of its key functionalities:

- **Run Package Binaries:** You can use npx to directly run executable files (binaries) provided by npm packages. These binaries are often included in the `package.json` file under the `bin` property.

- **Temporary Installation:** Unlike `npm install`, which installs a package globally or locally in your project's `node_modules` directory, npx executes the package without permanent installation. This is useful for one-time use cases or trying out packages before committing to a full installation.
- **Dependency Resolution:** If the package you want to run has dependencies (other packages it relies on), npx will automatically download and install them temporarily to ensure the executable can run successfully.
- **Avoids Version Conflicts:** By not globally installing packages, npx helps prevent potential version conflicts that might arise if you have multiple projects requiring different versions of the same package.

Here are some common use cases for npx:

- **Running Create React App:** You can use `npx create-react-app my-app` to quickly set up a new React project without globally installing `create-react-app`.
- **Executing CLI Tools:** Many npm packages provide command-line interface (CLI) tools. `npx` lets you run these tools directly, such as `npx prettier --write myfile.js` to format your JavaScript code using Prettier without installing it globally.
- **Testing Packages:** You can use `npx` to try out a package before installing it. For example, `npx my-package --help` might display the help documentation of a package named `my-package`.

In essence, npx is a convenient tool for running commands from npm packages without cluttering your system with global installations. It promotes a clean development workflow and helps you manage package dependencies effectively.

---

why is node js necessary to execute tailwind

While Tailwind CSS itself doesn't directly require Node.js to function, using Tailwind effectively typically involves Node.js for a few reasons:

1. Package Management and Installation:

- Tailwind CSS is distributed as an npm package. Node.js, along with its package manager (npm), is the standard way to install and manage JavaScript packages in your project. npm helps you download Tailwind and its dependencies efficiently.

2. PostCSS Integration:

- Tailwind CSS works best alongside PostCSS, a CSS processing tool. While Tailwind doesn't strictly require PostCSS, it leverages PostCSS plugins for functionalities like vendor prefixing (ensuring compatibility with older browsers) and optimizing the generated CSS. Node.js allows you to install and configure PostCSS and its plugins along with Tailwind.

3. Build Process Automation:

- In modern web development workflows, tasks like compiling Tailwind classes, minifying CSS, and combining multiple CSS files are often automated using build tools like Gulp or Webpack. These tools typically rely on Node.js to run and execute these automation tasks.

4. Standalone Option (Limited):

- It's important to note that Tailwind offers a standalone CLI (Command Line Interface) option that allows using it without Node.js. However, this approach has limitations. You can't leverage the full power of configuration files (`tailwind.config.js`) and might miss out on some functionalities compared to the Node.js-based workflow.

In summary, while Tailwind CSS itself can technically function without Node.js, Node.js provides a more robust and efficient development experience. It streamlines package management, PostCSS integration, and build process automation, making Tailwind easier to use and manage in your projects.

---

vite :

Vite is a relatively new and popular frontend build tool that offers a faster and more streamlined development experience compared to traditional tools like Webpack.

---

npx tailwindcss init

The command `npx tailwindcss init` is used to initialize Tailwind CSS in your project. Here's a breakdown of what it does:

1. Installs Dependencies (Optional):

- If you don't have Tailwind CSS and its dependencies installed in your project, `npx tailwindcss init` will take care of that for you. It uses npm (Node Package Manager) to download the necessary packages from the npm registry.

2. Creates Configuration File:

- The main purpose of `npx tailwindcss init` is to create a configuration file named `tailwind.config.js` at the root of your project. This file allows you to customize

Tailwind CSS to your specific needs.

- Default Configuration: By default, the generated `tailwind.config.js` file will be a minimal configuration with an empty structure. This means you'll need to add your desired customizations later.

- Full Configuration (Optional Argument): If you want a more complete starting point, you can use the optional argument `--full` with the command. This will create a `tailwind.config.js` file with the default Tailwind configuration settings included.

3. Integrates with PostCSS (Optional):

- While not strictly required for basic functionality, Tailwind CSS works best alongside PostCSS, a CSS processing tool. `npx tailwindcss init` doesn't directly install or configure PostCSS, but it might provide instructions or references in the terminal output to guide you through the integration process.

In essence, `npx tailwindcss init` sets the stage for using Tailwind CSS in your project by installing dependencies (if needed) and creating the essential configuration file.

Here are some additional points to consider:

- **Existing Project:** If you're running `npx tailwindcss init` in an existing project, make sure you have Node.js and npm installed on your system.
- **Customization:** After running the command, you'll likely need to edit the `tailwind.config.js` file to customize Tailwind CSS for your project. This might involve specifying the paths to your CSS files, enabling desired utility classes, or configuring other options.
- **PostCSS Integration:** For optimal functionality, you'll need to configure PostCSS and its plugins to work with Tailwind CSS. Documentation and tutorials are readily available online to guide you through this process.

Overall, `npx tailwindcss init` is a handy way to kickstart your Tailwind CSS project. By understanding its purpose and the potential next steps, you can effectively leverage Tailwind's utility classes to style your web pages efficiently.

---

npm install -D tailwindcss postcss autoprefixer :

The command `npm install -D tailwindcss postcss autoprefixer` uses npm (Node Package Manager) to install three packages for your project:

- **tailwindcss:** This is the main package, providing the utility classes and functionality for styling your web pages using Tailwind CSS.

- **postcss:** This is a CSS processing tool that acts as a middleman between your written CSS and the final output. It allows you to use plugins like Tailwind CSS to transform or enhance your CSS code.
- **autoprefixer:** This is a PostCSS plugin that automatically adds vendor prefixes to your CSS styles. Vendor prefixes are variations of CSS properties used to ensure compatibility with different web browsers.

Here's a breakdown of the flags used in the command:

- **-D:** This flag stands for "dev dependency". It tells npm to install the packages as development dependencies. This means these packages are not required for the final production build of your application but are necessary for the development process.

Why install these together?

- Tailwind CSS is designed to work with PostCSS. It utilizes PostCSS plugins to generate the final CSS styles based on the utility classes you use.
- Autoprefixer is a valuable plugin for PostCSS. It saves you time and effort by automatically adding vendor prefixes to your CSS, ensuring your styles work across different browsers.

Benefits of using this command:

- **Efficient Styling:** Tailwind CSS provides a collection of pre-built utility classes, allowing you to style your web pages quickly and efficiently.
- **Modern CSS Features:** PostCSS lets you leverage modern CSS features even if they're not yet supported by all browsers (autoprefixer handles compatibility).
- **Clean and Maintainable Code:** Tailwind CSS promotes writing clean and readable CSS by using clear and concise utility classes.
- **Browser Compatibility:** Autoprefixer ensures your styles work across different browsers by adding vendor prefixes.

In essence, this command installs the essential tools for using Tailwind CSS effectively in your project. It allows you to write modern CSS with utility classes while maintaining compatibility with various browsers.

After running this command:

- You'll likely need to configure Tailwind CSS in a file named `tailwind.config.js`. This file allows you to customize Tailwind's behavior and define which utility classes you want to use in your project.
- You might also need to set up PostCSS and its plugins (including autoprefixer) to work with your project's build process. This typically involves adding configuration to tools like Gulp or Webpack.

There are many resources available online to guide you through the configuration process for Tailwind CSS, PostCSS, and autoprefixer.

---

tailwind.config.js :

```
{
"scripts": {
"start": "vite"
},
"devDependencies": {
"autoprefixer": "^10.4.19",
"postcss": "^8.4.38",
"tailwindcss": "^3.4.3"
},
"dependencies": {
"vite": "^5.2.8"
}
}
```

Here's a breakdown of the key parts:

- **scripts**: This section defines npm commands that can be executed via the command line. In this case, you have a script named "start" which runs Vite.
- **devDependencies**: These are dependencies that are only needed for development purposes. They won't be included in the production build of your project. Here, you have three devDependencies:
  - **autoprefixer**: A PostCSS plugin to parse CSS and add vendor prefixes to CSS rules using values from the Can I Use website.
  - **postcss**: A tool for transforming CSS with JavaScript plugins. Autoprefixer is one such plugin.
  - **tailwindcss**: A utility-first CSS framework for rapidly building custom designs. It provides low-level utility classes to create custom designs quickly.
- **dependencies**: These are dependencies that are required for your project to run in production. Here, you have one dependency:
  - **vite**: A next-generation frontend build tool that significantly improves the frontend development experience. It's used to build modern web applications.
