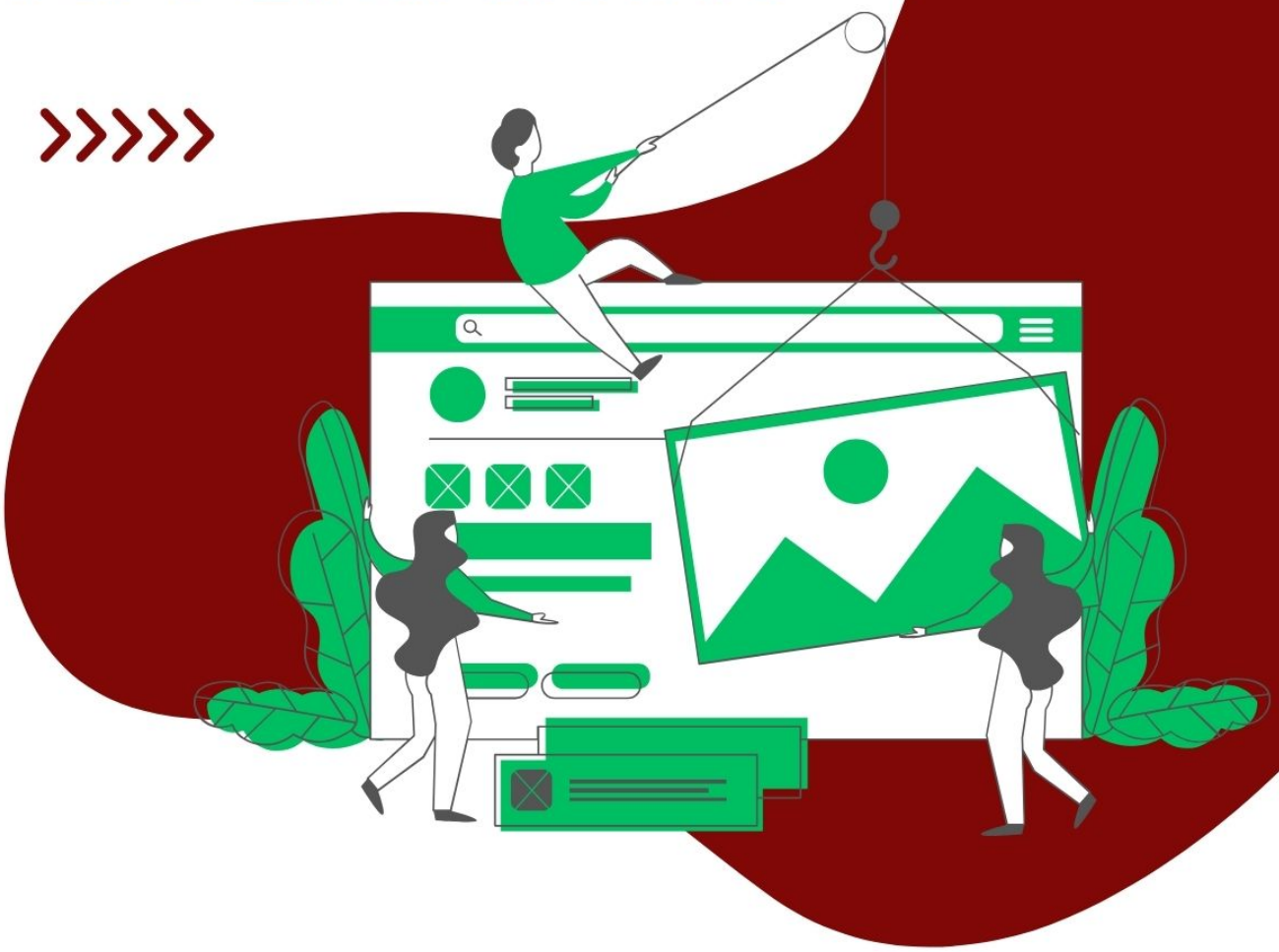
ADVANCED WEB APPLICATION



Prepared by:

Anthony S. Gacis, MIT

Instructor I

Table of Contents

Module 1 - Introduction to Front-end Development

Front-end development is the practice of building the user-facing elements of a website or web application. This includes the HTML, CSS, and JavaScript that create the structure, style, and interactivity of the web page.

In recent years, there has been a significant shift in the front-end development landscape. The rise of new technologies, such as React, Vue, and Svelte, has made it possible to build more dynamic and interactive web applications than ever before.

These technologies are all based on the concept of reactivity, which means that the user interface can automatically update whenever the underlying data changes. This makes it possible to create fluid and responsive web applications that provide a great user experience.

In this module, we will introduce you to the fundamentals of front-end development, with a focus on reactivity and the modern web. We will cover the following topics:

- Understanding the enhancement of modern web with javascript
- Reactive programming with Vue
- Bundling assets and optimization with Vite
- State management and routing

Front-end Development

Front-end development is the process of building the user-facing elements of a website or web application. This includes the HTML, CSS, and JavaScript that create the structure, style, and interactivity of the web page.

Front-end developers are responsible for creating a user interface that is both visually appealing and easy to use. They work closely with designers and back-end developers to ensure that the website or application meets the needs of the users.

The Importance of Front-End Development

Front-end development is an essential part of the web development process. A well-designed and developed front-end can make a big difference in the success of a website or application.

A good front-end can:

- Improve the user experience: A well-designed front-end can make it easier for users to find the information they need and complete tasks.
- Increase conversions: A visually appealing and user-friendly front-end can encourage users to stay on the website or application longer and make purchases.
- Boost SEO: A responsive and well-coded front-end can help a website rank higher in search engine results pages (SERPs).

The Core Technologies of Front-End Development

The core technologies of front-end development are HTML, CSS, and JavaScript.

- HTML (HyperText Markup Language) is used to define the structure of a web page.
- CSS (Cascading Style Sheets) is used to style the web page.
- JavaScript is used to add interactivity to the web page.

Front-end developers use these technologies to create web pages that are both visually appealing and functional.

Modern Front-End Development

In recent years, there has been a significant shift in the front-end development landscape. The rise of new technologies, such as React, Vue, and Svelte, has made it possible to build more dynamic and interactive web applications than ever before.

These technologies are all based on the concept of reactivity, which means that the user interface can automatically update whenever the underlying data changes. This makes it possible to create fluid and responsive web applications that provide a great user

experience.

In addition to reactive technologies, front-end developers are also using modern CSS frameworks, such as Bootstrap and Tailwind CSS, to create responsive and mobile-friendly web applications.

The Skills of a Front-End Developer

Front-end developers need to have a strong understanding of HTML, CSS, and JavaScript. They also need to be familiar with modern front-end technologies, such as React, Vue, and Svelte.

In addition to technical skills, front-end developers also need to have good design skills and be able to think creatively. They need to be able to understand the needs of the users and design a user interface that is both visually appealing and easy to use.

The Future of Front-End Development

The future of front-end development is bright. As new technologies emerge and existing technologies continue to improve, front-end developers will be able to create even more dynamic and interactive web applications.

Here are some of the trends that are shaping the future of front-end development:

- The rise of artificial intelligence (AI): AI is being used to develop new tools and technologies that can help front-end developers to be more productive and efficient.
- The increasing popularity of progressive web apps (PWAs): PWAs are web applications that can be installed on a user's device and provide a native app-like experience. Front-end developers are increasingly being asked to build PWAs.
- The growing importance of accessibility: Front-end developers are increasingly being asked to build web applications that are accessible to users with disabilities.

Front-end Frameworks

Front-end frameworks are essential tools for developing user interfaces (UIs) for web applications. They provide a structured and standardized approach to building UI components, making it easier to create consistent and maintainable code. For BSIT students, it is crucial to understand the different front-end frameworks available and their suitability for various types of projects.

Popular Front-end Frameworks

Several front-end frameworks have gained popularity among professionals. Each framework has its strengths, weaknesses, and target audience. Here's an overview of some of the most widely used frameworks:

- **React**: React is a JavaScript library for building user interfaces. It is known for its component-based approach, making it easy to create modular and reusable UI components. React is widely used for developing Single-Page Applications (SPAs) and dynamic web applications.

- **Vue.js**: Vue.js is another popular JavaScript framework for building UIs. It is similar to React in many ways but offers a more lightweight and approachable syntax. Vue.js is well-suited for smaller projects and applications that require a quick development turnaround.

- **Angular**: Angular is a TypeScript-based framework developed by Google. It provides a more comprehensive and opinionated approach to front-end development, offering a built-in dependency injection system, routing, and form handling. Angular is suitable for large-scale enterprise applications.

- **Svelte**: Svelte is a relatively new JavaScript framework that has gained traction in recent years. It differs from React and Vue.js in its compile-time reactivity, which means the framework handles reactivity more efficiently, leading to faster performance. Svelte is a promising choice for building performant web applications.

Factors to Consider When Choosing a Front-end

Framework

Several factors should be considered when choosing a front-end framework:

- **Project requirements**: The specific requirements of the project should dictate the choice of framework. For instance, if the project involves building a complex SPA, React or Angular might be suitable. For smaller projects or rapid prototyping, Vue.js or Svelte could be better choices.

- **Developer experience**: The framework's learning curve and ease of use should be considered. You should choose a framework that aligns with your experience level and learning preferences.

- **Community and support**: A large and active community around the framework can provide valuable resources, documentation, and support during development.

Setting Up Environment

Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, C#, Java, Python, PHP, Go, .NET). To download, just visit this URL https://code.visualstudio.com

Extensions

Icons

You can install Material Icon Theme it supports almost every file extension and can be customized for everyone's needs.

Auto Rename Tag

Automatically rename paired HTML/XML tag, same as Visual Studio IDE does.

Auto Close Tag

Automatically add HTML/XML close tag, same as Visual Studio IDE or Sublime Text does.

Code Spell Checker

A basic spell checker that works well with code and documents. The goal of this spell checker is to help catch common spelling errors while keeping the number of false positives low.

Error Lens

ErrorLens turbocharges language diagnostic features by making diagnostics stand out more prominently, highlighting the entire line wherever a diagnostic is generated by the language and also prints the message inline.

ESLint

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs.

GitLens

GitLens supercharges Git inside VS Code and unlocks untapped knowledge within each repository. It helps you to visualize code authorship at a glance via Git blame annotations and CodeLens, seamlessly navigate and explore Git repositories, gain valuable insights via rich visualizations and powerful comparison commands, and so much more.

Import Cost

This extension will display inline in the editor the size of the imported package. The extension utilizes webpack in order to detect the imported size.

Prettier - Code Formatter

Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.

Vue - Official Extension

The extension provides syntax highlighting, TypeScript support, and intellisense for template expressions and component props.

Module 2 - Introduction to Vite

Why Vite?

The Problems

Before ES modules were available in browsers, developers had no way to write JavaScript in a modularized fashion. This is why bundling tools were created. These tools crawl, process, and concatenate our source modules into files that can run in the browser.

Over time, tools like webpack, Rollup, and Parcel have greatly improved the development experience for frontend developers. However, as we build more and more complex applications, the amount of JavaScript we use is also increasing dramatically. It is not uncommon for large-scale projects to contain thousands of modules.

This is starting to cause performance bottlenecks for JavaScript-based tooling. It can often take an unreasonably long time (sometimes up to minutes!) to spin up a dev server, and even with Hot Module Replacement (HMR), file edits can take a couple of seconds to be reflected in the browser. This slow feedback loop can greatly affect developers' productivity and happiness.

Vite aims to address these issues by taking advantage of two new advancements in the ecosystem:

1. The availability of native ES modules in the browser.
2. The rise of JavaScript tools written in compile-to-native languages.

This allows Vite to be much faster than previous bundling tools, especially for large-scale projects.

Slow Server Start

When you start the development server for the first time, a traditional bundler has to crawl through and build your entire application before it can start serving it. This can be slow, especially for large projects with many dependencies.

Vite improves the development server start time by dividing the modules in your application into two categories: **dependencies** and **source code**.

- **Dependencies** are mostly plain JavaScript that doesn't change often during development. Vite pre-bundles dependencies using esbuild, which is a very fast tool

written in Go. This means that Vite can start serving your application much faster than a traditional bundler.
- **Source code** often contains non-plain JavaScript that needs to be transformed (e.g. JSX, CSS, or Vue/Svelte components). It also changes more often than dependencies. Vite serves source code over native ESM, which means that the browser can help with some of the bundling work. This makes Vite even faster for development, especially for large projects with a lot of source code.

Slow Updates

Traditional bundlers are inefficient at rebuilding the bundle when a file is edited. This is because they have to rebuild the entire bundle, even if only a small part of the application has changed. This can lead to slow update speeds, especially for large applications.

Some bundlers support Hot Module Replacement (HMR), which allows modules to be replaced without reloading the entire page. This can improve the development experience, but HMR update speed can still deteriorate as the size of the application grows.

Vite uses native ESM to perform HMR. This means that Vite can only invalidate the chain between the edited module and its closest HMR boundary, which is usually just the module itself. This makes HMR updates consistently fast, regardless of the size of the application.

Vite also leverages HTTP headers to speed up full page reloads. This means that Vite can let the browser do more of the work, such as caching dependency modules. This makes Vite even faster for development, especially for large applications.

Overall, Vite is a much faster and more efficient bundler than traditional bundlers. This makes it a great choice for developing large and complex web applications.

Vite as a bundler

Overview

Vite (French word for "quick", pronounced /vit/, like "veet") is a new build tool that aims to make web development faster and easier. It has two main parts:

- A development server that provides rich features over native ES modules, such as extremely fast Hot Module Replacement (HMR).
- A build command that bundles your code with Rollup to produce highly optimized static assets for production.

Vite is opinionated and comes with sensible defaults out of the box, but it is also highly extensible via plugins and a JavaScript API with full typing support.

Node Version Manager

Node Version Manager (NVM) is a tool for managing multiple versions of Node.js on Windows. It allows you to easily install, switch between, and uninstall different versions of Node.js, making it a valuable tool for JavaScript developers.

We will be utilizing Node Version Manager (NVM) to effectively manage and switch between different versions of the Node.js runtime environment.

To install, just visit https://github.com/coreybutler/nvm-windows

After you install nvm, open a terminal then run

```
nvm -v
```

You should see the version of nvm, if not then verify your installation.

To check the list of currently install node versions, just run `nvm list`

```
> nvm list
```

This will display all available versions, for first time user you will see an empty records.

To install a specific node version, just run `nvm install <specific version number>`

```
> nvm install 20.9.0
Downloading node.js version 20.9.0 (64-bit)...
Extracting node and npm...
Complete
npm v10.1.0 installed successfully.

Installation complete. If you want to use this version, type

nvm use 20.9.0
```

After executing the command, it will install that version of node.js. To verify just execute again the command `nvm list`

```
> nvm list

    20.9.0
```

However, it is currently not activated. To activate, just run `nvm use <install version number>`

```
> nvm use 20.9.0
Now using node v20.9.0 (64-bit)

> nvm list

  * 20.9.0 (Currently using 64-bit executable)
```

You will notice an asterisk (*) before the version number. This indicates that the version is currently being used.

To verify, just run `node -v` to check the node version.

```
> node -v
v20.9.0
```

Module 3 - Introduction to Single Page Application (SPA) with Frontend Frameworks

In today's web development landscape, single-page applications (SPAs) have become increasingly popular due to their ability to provide a more responsive and user-friendly experience. SPAs, unlike traditional multi-page applications (MPAs), load a single HTML document and dynamically update the page content using JavaScript, eliminating the need for full page reloads. This results in a smoother user experience, similar to that of native mobile applications.

To develop SPAs effectively, frontend frameworks have emerged as essential tools. These frameworks provide a structured approach to building complex client-side applications, simplifying the development process and enhancing maintainability. Popular frontend frameworks include React, Angular, and Vue.js.

Core Concepts of SPAs

Understanding the core concepts of SPAs is crucial for building effective web applications. Key concepts include:

1. **Client-side rendering**: SPAs render the user interface on the client-side using JavaScript, reducing the load on the server and improving responsiveness.
2. **Single HTML document**: SPAs load a single HTML document and dynamically update the content, eliminating page reloads and creating a seamless user experience.
3. **JavaScript routing**: SPAs utilize JavaScript routing to handle navigation between different sections of the application without reloading the page.
4. **Data fetching and manipulation**: SPAs fetch and manipulate data using JavaScript APIs, such as Fetch or Axios, to update the application's state and render the appropriate UI components.

Benefits of SPAs

SPAs offer several advantages over traditional MPAs, including:

1. **Enhanced user experience**: SPAs provide a more responsive and fluid user experience, similar to native mobile applications.
2. **Faster page load times**: By eliminating full page reloads, SPAs can significantly reduce page load times, improving user engagement.
3. **Improved scalability**: SPAs can handle large amounts of data and complex user interactions efficiently.
4. **Richer interactions**: SPAs enable more interactive and dynamic user experiences, such as real-time data updates and animations.

Frontend Frameworks for SPA Development

Frontend frameworks have become indispensable tools for developing SPAs. They provide a structured approach to building complex client-side applications, offering several benefits:

1. **Component-based architecture**: Frameworks promote component-based development, breaking down applications into reusable and maintainable components.
2. **Data binding**: Frameworks facilitate data binding between application state and the UI, ensuring consistent updates across the application.
3. **Routing and navigation**: Frameworks provide built-in routing mechanisms for handling navigation between different application sections.
4. **Tooling and ecosystem**: Frameworks offer rich ecosystems of tools and libraries,

simplifying development and maintenance.

Popular Frontend Frameworks for SPAs

Several frontend frameworks have gained popularity for SPA development, each with its strengths and characteristics:

1. **React**: Developed by Facebook, React is a JavaScript library for building user interfaces. Its component-based approach and virtual DOM make it efficient for rendering large applications.
2. **Angular**: Created by Google, Angular is a TypeScript-based framework for building scalable SPAs. It provides a comprehensive solution with features like dependency injection, routing, and two-way data binding.
3. **Vue.js**: Developed by Evan You, Vue.js is a lightweight and progressive JavaScript framework. Its simplicity and flexibility make it a popular choice for both beginners and experienced developers.

Frontend Exploration

React

What is React?

React is a JavaScript library for building user interfaces (UIs). It is declarative, efficient, and flexible, making it a popular choice for developing single-page applications (SPAs). React was created by Jordan Walke, a software engineer at Facebook, and is maintained by Facebook and a community of individual developers and companies.

Describing the UI

UI is built from small units like buttons, text, and images. React lets you combine them into reusable, nestable components. From websites to phone apps, everything on the screen can be broken down into components. In this chapter, you'll learn to create, customize, and conditionally display React components.

Your first component

React applications are built from isolated pieces of UI called components. A React component is a JavaScript function that you can sprinkle with markup. Components can be as small as a button, or as large as an entire page. Here is a Gallery component rendering three Profile components:

```
function Profile() {
  return (
    <img
      src="https://i.imgur.com/MK3eW3As.jpg"
      alt="Katherine Johnson"
    />
  );
}

export default function Gallery() {
  return (
    <section>
      <h1>Amazing scientists</h1>
      <Profile />
      <Profile />
      <Profile />
    </section>
  );
}
```

Importing and exporting components

You can declare many components in one file, but large files can get difficult to navigate. To solve this, you can export a component into its own file, and then import that component from another file:

```
// Gallery.js
import Profile from './Profile.js';

export default function Gallery() {
  return (
    <section>
      <h1>Amazing scientists</h1>
      <Profile />
      <Profile />
      <Profile />
    </section>
  );
}
```

```
// Profile.js
export default function Profile() {
    return (
        <img
            src="https://i.imgur.com/QIrZWGIs.jpg"
            alt="Alan L. Hart"
        />
    );
}
```

Vue

What is Vue

Vue (pronounced /vju:/, like view) is a JavaScript framework for building user interfaces. It builds on top of standard HTML, CSS, and JavaScript and provides a declarative and component-based programming model that helps you efficiently develop user interfaces, be they simple or complex.

Here is a minimal example:

```
import { createApp, ref } from 'vue'

createApp({
  setup() {
    return {
      count: ref(0)
    }
  }
}).mount('#app')

<div id="app">
    <button @click="count++">
        Count is: {{ count }}
    </button>
</div>
```

The above example demonstrates the two core features of Vue:

- **Declarative Rendering**: Vue extends standard HTML with a template syntax that

allows us to declaratively describe HTML output based on JavaScript state.

- **Reactivity**: Vue automatically tracks JavaScript state changes and efficiently updates the DOM when changes happen.

The Progressive Framework

Vue is a framework and ecosystem that covers most of the common features needed in frontend development. But the web is extremely diverse - the things we build on the web may vary drastically in form and scale. With that in mind, Vue is designed to be flexible and incrementally adoptable. Depending on your use case, Vue can be used in different ways:

- Enhancing static HTML without a build step
- Embedding as Web Components on any page
- Single-Page Application (SPA)
- Fullstack / Server-Side Rendering (SSR)
- Jamstack / Static Site Generation (SSG)
- Targeting desktop, mobile, WebGL, and even the terminal

Single-File Components

In most build-tool-enabled Vue projects, Vue components using an HTML-like file format called Single-File Component (also known as *.vue files, abbreviated as SFC). A Vue SFC, as the name suggests, encapsulates the component's logic (JavaScript), template (HTML), and styles (CSS) in a single file. Here's the previous example, written in SFC format:

```
<script setup>
import { ref } from 'vue'
const count = ref(0)
</script>

<template>
  <button @click="count++">Count is: {{ count }}</button>
</template>

<style scoped>
button {
  font-weight: bold;
}
</style>
```

SFC is a defining feature of Vue and is the recommended way to author Vue components if your use case warrants a build setup.

Svelte

What is Svelte

Svelte is a tool for building web applications. Like other user interface frameworks, it allows you to build your app declaratively out of components that combine markup, styles and behaviours.

These components are compiled into small, efficient JavaScript modules that eliminate overhead traditionally associated with UI frameworks.

You can build your entire app with Svelte (for example, using an application framework like SvelteKit), or you can add it incrementally to an existing codebase. You can also ship components as standalone packages that work anywhere.

Your first component

In Svelte, an application is composed from one or more components. A component is a reusable self-contained block of code that encapsulates HTML, CSS and JavaScript that belong together, written into a .svelte file.

Adding data

A component that just renders some static markup isn't very interesting. Let's add some data.

First, add a script tag to your component and declare a name variable:

```
// App.svelte
<script>
    let name = 'Svelte';
</script>
<h1>Hello World!</h1>
```

Then, we can refer to name in the markup:

```
// App.svelte
<h1>Hello {name}!</h1>
```

Inside the curly braces, we can put any JavaScript we want. Try changing name to name.toUpperCase() for a shoutier greeting.

```
// App.svelte
<h1>Hello {name.toUpperCase()}!</h1>
```

UI Development with Vue

Creating your first project

First you need to install yarn, I recommend this for faster package resolution.

```
npm install --global yarn
```

Once installed, create a vue project via vite.

```
> yarn create vite

yarn create v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Installed "create-vite@5.0.0" with binaries:
      - create-vite
      - cva
√ Project name: ... vue-project
√ Select a framework: » Vue
√ Select a variant: » JavaScript

Scaffolding project in C:\Users\Admin\Desktop\vue-project...

Done. Now run:

  cd vue-project
  yarn
  yarn dev

Done in 210.31s.
```

Make sure to enter your project name, vue as a framework and Javascript as a variant. Then run the following commands

```
> cd vue-project
> yarn
> yarn dev
```

Inspecting your project

- node_modules
- public
- src
- .gitignore
- index.html
- package.json
- README.md
- vite.config.js
- yarn.lock

node_modules: This directory contains all the third-party dependencies installed for your Vue project. It's managed by Node Package Manager (npm) or Yarn and should not be modified manually.

public: This directory holds static assets like images, fonts, and other files that will be directly served by the web server. These files are not bundled into the main JavaScript application.

src: This directory contains the source code for your Vue application. It includes components, router configuration, store, and other JavaScript files that define the functionality and behavior of your app.

.gitignore: This file specifies which files and directories should be ignored by Git version control. It prevents unnecessary files from being tracked and avoids clutter in your repository.

index.html: This file serves as the entry point for your web application. It defines the basic HTML structure of your app and includes the main JavaScript bundle generated by Vue.

package.json: This file contains metadata about your project, including its name, version, dependencies, and scripts. It's used by npm and Yarn to manage dependencies and run project commands.

README.md: This file provides documentation about your project, explaining its purpose, usage, and installation instructions. It serves as a reference for new contributors and users.

vite.config.js: This file configures Vite, the Vue build tool, for your project. It specifies

options like server settings, file processing, and dependency resolution.

yarn.lock: This file is generated by Yarn and contains a lockfile that ensures consistent dependencies across different environments. It prevents dependency conflicts and ensures the same versions of libraries are used.

Basic syntax of Vue

First let's delete everything inside src folder except the **App.vue**

The App.vue

Delete the content and insert the following code:

```
<template>
</template>
```

This template tag is where our html code resides.

Now add script tag with setup property above it.

```
+ <script setup>
+ </script>
<template>
</template>
```

The `<script setup>` tag is a new syntax introduced in Vue 3.3 for using the Composition API inside Single-File Components (SFCs). It is the recommended syntax if you are using both SFCs and Composition API.

Conditional Rendering

The directive v-if is used to conditionally render a block. The block will only be rendered if the directive's expression returns a truthy value.

For example,

```
<script setup>
</script>
<template>
+    <h1 v-if="true">True</h1>
</template>
```

You can use the v-else directive to indicate an "else block" for v-if:

```
<script setup>
</script>
<template>
    <h1 v-if="true">True</h1>
+    <h1 v-else>False</h1>
</template>
```

The v-else-if, as the name suggests, serves as an "else if block" for v-if. It can also be chained multiple times:

```
<script setup>
</script>
<template>
    <h1 v-if="true">True</h1>
+    <h1 v-else-if="true & true">True & True</h1>
+    <h1 v-else-if="true & false">True & False</h1>
    <h1 v-else>False</h1>
</template>
```

Another option for conditionally displaying an element is the v-show directive. The usage is largely the same:

```
<script setup>
</script>
<template>
    <h1 v-if="true">True</h1>
    <h1 v-else-if="true & true">True & True</h1>
    <h1 v-else-if="true & false">True & False</h1>
    <h1 v-else>False</h1>
+   <h1 v-show="true">Using Show</h1>
</template>
```

v-if vs v-show

v-if is "real" conditional rendering because it ensures that event listeners and child components inside the conditional block are properly destroyed and re-created during toggles.

v-if is also lazy: if the condition is false on initial render, it will not do anything - the conditional block won't be rendered until the condition becomes true for the first time.

In comparison, v-show is much simpler - the element is always rendered regardless of initial condition, with CSS-based toggling.

Generally speaking, v-if has higher toggle costs while v-show has higher initial render costs. So prefer v-show if you need to toggle something very often, and prefer v-if if the condition is unlikely to change at runtime.

List Rendering

We can use the v-for directive to render a list of items based on an array. The v-for directive requires a special syntax in the form of item in items, where items is the source data array and item is an alias for the array element being iterated on:

```
<script setup>
</script>
<template>
    //           item in     items
    <li v-for="number in [1, 2, 3, 4]">
        {{ number }}
    </li>
</template>
```

Single File Component

Inside src folder, create another file named `BaseButton.vue` then insert the following code. It is recommended to name your component with multi-word.

```
// BaseButton.vue
<script setup>
</script>
<template>
    <button>Click me</button>
</template>
```

Then in App.vue, import the newly created component

```
// App.vue
<script setup>
    import BaseButton from './BaseButton.vue'
</script>
<template>
    <BaseButton></BaseButton>
</template>
```

This will import our component.

Reactivity

Using `ref` Function in Vue with `script setup`

The ref function is an essential tool in the Composition API when working with script setup in Vue. It allows you to create reactive references for various data types, including DOM elements, reactive objects, and primitive values.

Declaring and Initializing a ref:

1. **Import**: Start by importing the ref function from Vue:

```
<script setup>
import { ref } from 'vue';
</script>
```

2. **Declare and Define**: Use ref with the desired initial value:

```
const myRef = ref(42); // Declare a ref with initial value of 42
const textRef = ref('Initial text'); // Ref for a string
```

Accessing ref Values:

- **Template**: Inside the template, you can directly access the ref's unwrapped value using its name:

```
<template>
  <p>{{ myRef }}</p>
</template>
```

- **Script**: Within script setup, access the value with .value:

```
<script setup>
const double = () => myRef.value * 2;
</script>
```

Updating ref Values:

- **Template**: Use the v-model directive to bind a ref to a DOM input:

```
<template>
    <input v-model="textRef" type="text"/>
</template>
```

- **Script**: Assign a new value directly to .value:

```
<script setup>
const updateValue = () => {
  myRef.value = 50;
};
</script>
```

Template Refs:

- Use ref with a template ref for DOM elements:

```
<template>
    <input ref="myInput" type="text"/>
</template>

<script setup>
const myInput = ref(null);
// Access DOM element properties
myInput.value.focus();
</script>
```

Using Vue Router

Vue Router is a popular routing library for Vue.js applications. It allows you to easily manage navigation between different views in your application.

Example

In Vue 3 using script setup, it's recommended to create a separate router.js file to manage the router configuration. This approach promotes code organization and separation of concerns, making the main.js file more concise and focused on initializing the Vue application.

1. Create a separate router.js file inside `src` directory, then copy and paste the following code.

```
import { createRouter, createWebHistory } from  'vue-router';

const routes = [
  {
      path: '/',
      name: 'home',
      component: () => import('./components/Home.vue'),
  },
];

const router = createRouter({
  history: createWebHistory(),
  routes,
});

export default router;
```

This code defines a single route: '/' (home page). The component property for each route specifies the respective Vue component to render when that route is active.

2. Create a component named `Home.vue` inside components folder then paste the following code

```
<template>
    Home page
</template>
```

3. In main.js file, update the code using the following.

```
import { createApp } from 'vue'
import App from './App.vue'
import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap/js/index.umd.js'
+   import router from './router'

-   createApp(App).mount('#app')
+   let app = createApp(App)
+   app.use(router)
+   app.mount('#app')
```

4. In app.vue, update code using the following

```
<template>
    <router-view></router-view>
</template>
```

5. Run the app then check. It will load our Home.vue component when you access the root path that is '/'

Project Development

We are going to build a sample project using Vue. Before to do that, let's clear some codes. In App.vue, copy the following code snippet.

```
// App.vue
<script setup>
</script>
<template>
</template>
```

Then in template tag copy the following html templates

```
<div class="container py-5" style="height: 100vh;">
    <div class="row">
        <div class="col-md-4">
            <div class="card mt-3" style="width: 18rem;">
                <a href="javascript: void(0)" class="text-decoration-
none">
                    <div class="card-body">
                        <h5 class="card-title">Flexbox</h5>
                        <p class="card-text">A CSS layout module.</p>
                    </div>
                </a>
            </div>
        </div>
        <div class="col-md-8">
            <div class="form-group">
                <label>Child Count:</label>
                <input type="number" min="1" v-model="childCount"
class="form-control">
            </div>
            <div class="form-group">
                <label>Flex Direction:</label>
                <select class="form-select" v-model="flexDirection">
                    <option value="">-- Please select --</option>
                    <option value="flex-row">row</option>
                    <option value="flex-row-reverse">row-
reverse</option>
                    <option value="flex-column">column</option>
                    <option value="flex-column-reverse">column-
```

```
reverse</option>
                </select>
            </div>
            <div class="form-group">
                <label>Flex Wrap:</label>
                <select class="form-select" v-model="flexWrap">
                    <option value="">-- Please select --</option>
                    <option value="flex-nowrap">nowrap</option>
                    <option value="flex-wrap">wrap</option>
                    <option value="flex-wrap-reverse">wrap-
reverse</option>
                </select>
            </div>
            <div class="form-group">
                <label>Justify Content:</label>
                <select class="form-select" v-
model="flexJustifyContent">
                    <option value="">-- Please select --</option>
                    <option value="justify-content-start">flex-
start</option>
                    <option value="justify-content-end">flex-
end</option>
                    <option value="justify-content-
center">center</option>
                    <option value="justify-content-around">space-
around</option>
                    <option value="justify-content-evenly">space-
evenly</option>
                    <option value="justify-content-between">space-
between</option>
                </select>
            </div>
            <div class="form-group">
                <label>Align Items:</label>
                <select class="form-select" v-model="flexAlignItems">
                    <option value="">-- Please select --</option>
                    <option value="align-items-stretch">stretch</option>
                    <option value="align-items-
baseline">baseline</option>
                    <option value="align-items-center">center</option>
                    <option value="align-items-start">flex-
start</option>
                    <option value="align-items-end">flex-end</option>
                </select>
            </div>
            <div class="form-group">
```

```
                <label>Align Content:</label>
                <select class="form-select" v-model="flexContent">
                    <option value="">-- Please select --</option>
                    <option value="align-content-center">center</option>
                    <option value="align-content-start">flex-
start</option>
                    <option value="align-content-end">flex-end</option>
                    <option value="align-content-around">space-
around</option>
                    <option value="align-content-evenly">space-
evenly</option>
                    <option value="align-content-between">space-
between</option>
                </select>
            </div>
            <div class="mt-5">
                <div style="width: 100%; height: 400px;"
                    class="bg-primary bg-opacity-10 d-flex border
border-4 border-warning" :class="flexClass">
                    <div v-for="child in childCount" :key="child"
                        class="p-5 bg-primary text-center d-flex align-
items-center justify-content-center fs-1 text-white border border-2
border-dark">
                        {{ child }}
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

- The template starts with a container div with classes container, py-5, and an inline style setting the height to 100vh (full view height).
- Inside the container, there's a row div for horizontal layout with two child col divs.

Flexbox Controls:

- The first column (col-md-4) contains a card displaying information about Flexbox.
- The second column (col-md-8) holds several form elements for controlling various Flexbox properties:
- Child Count: Input field for specifying the number of child elements to display in the Flexbox container.
- Flex Direction: Select box for choosing the direction of the flex lines (row, column, etc.).
- Flex Wrap: Select box for setting how flex lines wrap (nowrap, wrap, etc.).

- Justify Content: Select box for controlling the alignment of flex items along the main axis.
- Align Items: Select box for controlling the alignment of flex items along the cross axis.
- Align Content: Select box for controlling the alignment of flex lines along the cross axis.

Flexbox Preview:

- Below the form is a large div with various inline styles:
- width: 100% and height: 400px for fixed size.
- bg-primary bg-opacity-10 for a subtle blue background.
- d-flex for enabling flexbox layout.
- border border-4 border-warning for a thick yellow border.
- :class="flexClass" dynamically applies additional classes based on the user's selections.
- Inside this div, a v-for loop iterates over the childCount and creates the specified number of child divs:
- Each child div has various styles for sizing, background, text, and borders.
- The content of each child displays its unique index using {{ child }}

Data and Reactivity:

- The template utilizes several Vue features for data binding and dynamic rendering:
- v-model directive binds the form elements to corresponding data properties (childCount, flexDirection, etc.).
- :class directive dynamically adds classes to the flexbox container based on the selected options.
- v-for directive iterates over the childCount and renders the corresponding number of child elements.

Next, inside `<script setup>` tag copy the following snippets

```
import { ref, computed } from 'vue'

const childCount = ref(1)
const flexDirection = ref('')
const flexWrap = ref('')
const flexJustifyContent = ref('')
const flexAlignItems = ref('')
const flexContent = ref('')

const flexClass = computed(() => {
    let classes = [
        flexDirection.value,
        flexWrap.value,
        flexJustifyContent.value,
        flexAlignItems.value,
        flexContent.value
    ]

    return classes.join(' ')
})
```

This Vue script defines the data and computed properties used by the previously added html template. Let's break down the code:

Imports:

- ref and computed functions are imported from vue.

Data Properties:

- childCount: ref variable holding the number of child elements (initially set to 1).
- flexDirection: ref variable holding the selected flex direction (initially empty).
- flexWrap: ref variable holding the selected flex wrap behavior (initially empty).
- flexJustifyContent: ref variable holding the selected justification option (initially empty).
- flexAlignItems: ref variable holding the selected alignment option for items (initially empty).
- flexContent: ref variable holding the selected alignment option for content (initially empty).

Computed Property:

- flexClass: computed property that dynamically generates the class string based on selected options.

`flexClass`

1. An array classes is created.
2. The current values of flexDirection, flexWrap, flexJustifyContent, flexAlignItems, and flexContent are added to the array.
3. The join(' ') method concatenates all elements in the array with a space between them, resulting in a single class string.
4. This class string is then used in the template to dynamically apply styles based on the user's selections.

Run the project using `yarn dev` then try the output.

Module 4 - Application Programming Interface

Application Programming Interfaces (APIs) have become indispensable tools for software development and integration. APIs provide a structured and standardized way for different applications to communicate and exchange data, enabling seamless integration and innovation across the digital landscape. This module delves into the fundamentals of APIs, exploring their concepts, functionalities, and real-world applications.

What is API?

An API (Application Programming Interface) is a set of rules and specifications that define how two software applications can communicate with each other. It acts as an intermediary, translating requests from one application into understandable instructions for the other, allowing them to exchange data and functionality. APIs are the backbone of modern software development, facilitating data sharing, integration, and innovation.

Key Characteristics of APIs

- **Modular Design**: APIs are designed in a modular fashion, encapsulating specific functionalities and making them accessible to external applications.
- **Standardized Communication**: APIs adhere to standardized protocols, such as REST or SOAP, ensuring consistent communication between different software systems.
- **Data Abstraction**: APIs abstract away the underlying implementation details, allowing developers to focus on the data they want to access or manipulate.

Types of APIs

- **Public APIs**: Public APIs are freely accessible and documented, allowing anyone to build applications that interact with the API's provided services or data.
- **Private APIs**: Private APIs are restricted to authorized users or applications within an organization or ecosystem.
- **Partner APIs**: Partner APIs are designed for specific partners or collaborators, enabling them to integrate their applications or services with the API provider's platform.

API Lifecycle

The API lifecycle encompasses the entire process of creating, maintaining, and evolving an API to ensure its effectiveness and relevance.

- **Design and Planning**: Defining the API's purpose, target audience, and functional scope.
- **Development and Implementation**: Creating the API's code, documentation, and testing infrastructure.
- **Deployment and Access**: Publishing the API for public or restricted access.
- **Maintenance and Evolution**: Monitoring API usage, addressing bugs, and implementing new features or enhancements.

Benefits of Utilizing APIs

- **Accelerated Development**: APIs provide pre-built functionalities, reducing development time and effort.
- **Enhanced Integration**: APIs enable seamless integration between different applications and systems.
- **Innovation and Collaboration**: APIs foster innovation by enabling developers to build upon existing services and data.
- **Scalability and Flexibility**: APIs facilitate scalable and flexible solutions that can adapt to changing requirements.

Using public API

JSONPlaceholder is a free online REST API that provides a vast collection of fake data for testing and prototyping purposes. It offers a variety of resources, including posts, comments, albums, photos, todos, and users, making it an ideal tool for practicing API interactions and data manipulation in JavaScript. This is the url https://jsonplaceholder.typicode.com/

Fetching Data with Fetch API

The Fetch API is a modern and commonly used method for making HTTP requests in JavaScript. It provides a Promise-based interface, making asynchronous data fetching more manageable. Here's an example of fetching posts from the JSONPlaceholder API using the Fetch API:

```
fetch('https://jsonplaceholder.typicode.com/posts')
    .then(response => response.json())
    .then(data =>
        console.log(data))
    .catch(error =>
        console.error(error)
    );
```

This code snippet initiates a GET request to the /posts endpoint of the JSONPlaceholder API. The response is then parsed as JSON data, and the resulting data object is logged to the console.

Creating, Updating, and Deleting Data

JSONPlaceholder also supports creating, updating, and deleting data. For instance, to create a new post, you can use the POST method and send the post data as JSON:

```
const newPost = {
    title: 'My New Post',
    body: 'This is the content of my new post.',
};

fetch('https://jsonplaceholder.typicode.com/posts', {
    method: 'POST',
    body: JSON.stringify(newPost),
    headers: {
        'Content-Type': 'application/json',
    },
})
    .then(response => response.json())
    .then(data =>
        console.log(data))
    .catch(error =>
        console.error(error));
```

Similarly, to update an existing post, you would use the PUT method and specify the ID of the post to update:

```
const updatedPost = {
    id: 1, // ID of the post to update
    title: 'Updated Title',
    body: 'Updated post content.',
};

fetch(`https://jsonplaceholder.typicode.com/posts/${updatedPost.id}`, {
    method: 'PUT',
    body: JSON.stringify(updatedPost),
    headers: {
        'Content-Type': 'application/json',
    },
})
    .then(response => response.json())
    .then(data =>
        console.log(data))
    .catch(error =>
        console.error(error)
    );
```

Finally, to delete a post, you can use the DELETE method and specify the ID of the post to

delete:

```
fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`, {
    method: 'DELETE',
})
    .then(response => console.log(response.status))
    .catch(error => console.error(error));
```

Module 5 - Deployment to Vercel from Github

Easy Steps to Deploy a Vue.js App to Vercel from GitHub:

1. Prerequisites:
    - A Vercel account. Sign up for free if you haven't already.
    - A Github account with your Vue.js project pushed to a repository.
    - Node.js and NPM installed on your machine.
2. Connect Vercel to Github:
    - Go to your Vercel account and click on "Import Project".
    - Choose "Import from Github".
    - Select the Github repository containing your Vue.js project and authorize Vercel to access it.
3. Configure Deployment Settings:
    - Vercel will automatically detect your Vue.js project and pre-configure the deployment settings.
    - You can review and adjust settings like the build command, environment variables, and custom domains (optional).
    - Make sure the "Build command" field is set to npm run build or yarn build depending on your project setup.
4. Deploy your Project:
    - Click the "Deploy" button.
    - Vercel will build your project and deploy it to a unique URL.
    - You can access the deployment URL in the Vercel dashboard.
5. Push Changes and Automate Deployments:
    - Any future changes pushed to your Github repository will automatically trigger a new deployment on Vercel.
    - Vercel also offers integrations with CI/CD tools like Travis CI and CircleCI for automated builds and deployments.