JavaScript and Vue.js: Definitive Guide from Beginner to Expert-Level Topics

JavaScript (JS) is a dynamic, event-driven programming language that powers the web, running in browsers and on servers via Node.js. Vue.js is a progressive JavaScript framework for building reactive, component-based user interfaces, known for its simplicity and flexibility. This guide is the ultimate resource for mastering JavaScript with a focus on Vue 3, covering every topic from beginner to expert level with exhaustive explanations, practical examples, edge cases, and cross-connections, matching the depth of a Rust README.

Table of Contents

1. Introduction to JavaScript and Vue.js
2. Getting Started: First Vue App
3. JavaScript Versions and Ecosystem
4. Basic JavaScript Concepts
    - Variables and Data Types
    - Functions and Arrow Functions
    - Control Flow
5. JavaScript Modules
6. Asynchronous JavaScript
7. Object-Oriented and Functional Programming
8. Vue.js Core Concepts
    - Composition API
    - Reactivity System
    - Components
    - Directives
9. Vue Router
10. State Management with Pinia
11. Advanced Vue Features
    - Suspense and Lazy Loading
    - Teleport
    - Render Functions and JSX
12. TypeScript with Vue
13. Performance Optimization

14. Security in Vue Apps
15. Testing Vue Apps
    - Vitest
    - Vue Test Utils
    - Cypress
16. Tooling and Ecosystem
    - Vite
    - Vue CLI
    - Nuxt.js
17. Sample Project: Task Management App
18. Resources

Introduction to JavaScript and Vue.js

JavaScript, created in 1995, is the backbone of web development, enabling interactivity in browsers and server-side logic via Node.js. Its key features include:

- **Dynamic Typing**: Variables can change types at runtime.
- **Event-Driven**: Handles asynchronous events (e.g., clicks, HTTP requests).
- **Cross-Platform**: Runs in browsers, servers, and desktops (Electron).
- **Use Cases**: Web apps, APIs, games, IoT, and mobile apps (React Native).

Vue.js, created by Evan You in 2014, is a lightweight framework for building reactive UIs. Vue 3 (released 2020) introduces:

- **Composition API**: Flexible, function-based component logic.
- **Reactivity**: Efficient, proxy-based data updates.
- **Ecosystem**: Vue Router, Pinia, Vite, Nuxt.js.
- **Use Cases**: Single-page apps (SPAs), progressive web apps (PWAs), and server-side rendering (SSR).

This guide focuses on Vue 3, leveraging modern JavaScript for scalable front-end development.

Getting Started: First Vue App

Install Node.js (v22.x LTS) from nodejs.org. Verify:

```
node -v
npm -v
```

Create a Vue project with Vite:

```
npm create vue@latest
# Follow prompts: enable TypeScript, Vue Router, Pinia, Vitest
cd my-vue-app
npm install
npm run dev
```

Edit `src/App.vue`:

```
<script setup>
import { ref } from 'vue';
const message = ref('Hello, Vue!');
</script>

<template>
  <h1>{{ message }}</h1>
  <button @click="message = 'Vue is awesome!'">Change</button>
</template>

<style scoped>
h1 { color: #42b983; }
</style>
```

Open `http://localhost:5173`. Use VS Code with the Vue extension (`Vue.volar`) or WebStorm for development.

JavaScript Versions and Ecosystem

JavaScript evolves via ECMAScript (ES) standards:

- **ES5 (2009)**: Baseline, widely supported.
- **ES6/ES2015**: Arrow functions, `let/const`, promises, modules.
- **ES2020**: Optional chaining (`?.`), nullish coalescing (`??`), BigInt.
- **ES2025**: Proposed `Array.prototype.at`, `Object.hasOwn`, record/tuple types.

Key tools:

- **Node.js**: Server-side runtime.
- **NPM/Yarn/PNPM**: Package managers.
- **Vite**: Fast build tool for Vue.
- **TypeScript**: Static types for JS.

Check Node version:

```
node -v
```

Use modern JS in Vue via Vite's ES modules support.

Basic JavaScript Concepts

Variables and Data Types

JavaScript is dynamically typed with primitives and objects:

- **Primitives**: `number`, `string`, `boolean`, `undefined`, `null`, `symbol`, `bigint`.
- **Objects**: Arrays, functions, `Map`, `Set`, custom objects.

```
let x = 42;
const name = 'Alice';
let obj = { id: 1, active: true };
let arr = [1, 2, 3];
let maybeNull = null;

console.log(`${name} has ${arr.length} items`); // Alice has 3 items
console.log(obj?.id ?? 'Unknown'); // 1
```

Edge Cases:

- **Type Coercion**: `'5' + 2 = '52'`; `'5' - 2 = 3`.
- **NaN**: `0 / 0` or invalid math.
- **Undefined vs. Null**: `undefined` for uninitialized; `null` for intentional absence.

Cross-Connections:

- **Reactivity**: Vue's `ref` wraps primitives.
- **TypeScript**: Enforce types.
- **Arrays**: Used in Vue's `v-for`.

Functions and Arrow Functions

Functions are first-class objects; arrow functions simplify syntax.

```
function add(a, b = 0) {
  return a + b;
}

const multiply = (x, y) => x * y;

const greet = (name) => `Hello, ${name}!`;

console.log(add(5, 3)); // 8
console.log(multiply(2, 4)); // 8
console.log(greet('Alice')); // Hello, Alice!
```

Advanced Features:

- **Rest Parameters**: `function sum(...nums)`.
- **Destructuring**: `({ x, y }) => x + y`.
- **IIFE**: `(function() { ... })()`.

Edge Cases:

- **this Binding**: Arrow functions inherit `this`; regular functions bind dynamically.
- **Default Parameters**: Evaluated at call time.

- **Hoisting**: Function declarations hoisted; expressions not.

Cross-Connections:

- **Vue Methods**: Define in `setup`.
- **Async**: Combine with `async`/`await`.
- **Event Handling**: Arrow functions in `@click`.

Control Flow

JavaScript supports `if`, `switch`, `for`, `forEach`, and loops.

```
const x = 7;
if (x > 5) {
  console.log('Greater');
} else {
  console.log('Lesser or equal');
}

const day = 'Monday';
switch (day) {
  case 'Monday':
    console.log('Start week');
    break;
  default:
    console.log('Other day');
}

const arr = [1, 2, 3];
for (const n of arr) {
  console.log(n);
}

arr.forEach(n => console.log(n * 2)); // 2, 4, 6
```

Advanced Features:

- **Optional Chaining**: `obj?.prop`.
- **Nullish Coalescing**: `x ?? defaultValue`.
- **Logical OR Short-Circuit**: `x || doSomething()`.

Edge Cases:

- **Falsy Values**: `0`, `''`, `null`, `undefined`, `NaN`.
- **Loop Mutation**: Modifying arrays during iteration.
- **Switch Fallthrough**: Missing `break` causes unexpected flow.

Cross-Connections:

- **Vue Directives**: `v-if`, `v-for`.
- **Reactivity**: Control flow in computed properties.

- **Error Handling**: Combine with `try`/`catch`.

JavaScript Modules

ES modules enable modular code.

```
// math.js
export const add = (a, b) => a + b;
export default (x, y) => x * y;

// main.js
import multiply, { add } from './math.js';
console.log(add(2, 3)); // 5
console.log(multiply(2, 3)); // 6
```

Advanced Features:

- **Dynamic Imports**: `const { fn } = await import('./module.js')`.
- **Tree Shaking**: Vite removes unused exports.
- **Top-Level Await**: `const data = await fetchData()`.

Edge Cases:

- **Circular Dependencies**: Cause runtime errors.
- **Default vs. Named**: Mixing can confuse imports.
- **CJS/ESM Interop**: Node.js dual-mode modules.

Cross-Connections:

- **Vue Components**: Imported in `script setup`.
- **Pinia**: Modular stores.
- **Vite**: Optimizes module bundling.

Asynchronous JavaScript

Handle async operations with promises, async/await, and event loops.

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve('Data loaded'), 1000);
  });
}

async function load() {
  try {
    const data = await fetchData();
    console.log(data); // Data loaded
  } catch (err) {
    console.error(err);
  }
}
```

```
load();
```

Advanced Features:

- **Promise.all**: `await Promise.all([p1, p2])`.
- **AbortController**: Cancel fetches.
- **Async Iterators**: `for await (const x of asyncGen())`.

```
const controller = new AbortController();
async function fetchWithTimeout(url, timeout = 5000) {
  const id = setTimeout(() => controller.abort(), timeout);
  const res = await fetch(url, { signal: controller.signal });
  clearTimeout(id);
  return res.json();
}
```

Edge Cases:

- **Uncaught Rejections**: Always use `try`/`catch`.
- **Memory Leaks**: Unresolved promises hold references.
- **Microtasks**: Promises resolve before timers.

Cross-Connections:

- **Vue Lifecycle**: Async setup in components.
- **Pinia**: Async actions.
- **Testing**: Mock async APIs.

Object-Oriented and Functional Programming

JavaScript supports OOP and functional paradigms.

```
// OOP
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    return `Hello, ${this.name}!`;
  }
}

const alice = new Person('Alice');
console.log(alice.greet()); // Hello, Alice!

// Functional
const add = (a, b) => a + b;
const compose = (f, g) => x => f(g(x));
const double = x => x * 2;
const square = x => x ** 2;
```

```
const doubleThenSquare = compose(square, double);
console.log(doubleThenSquare(3)); // 36
```

Advanced Features:

- **Prototypes**: `Object.create`, `__proto__`.
- **Closures**: Encapsulate state.
- **Immutability**: `Object.freeze`, spread operator.

Edge Cases:

- **this Binding**: Lost in extracted methods.
- **Prototype Pollution**: Modifying `Object.prototype`.
- **Pure Functions**: Avoid side effects in Vue reactivity.

Cross-Connections:

- **Vue Components**: Class-like with Composition API.
- **Reactivity**: Functional computed properties.
- **Pinia**: Functional stores.

Vue.js Core Concepts

Composition API

Vue 3's Composition API organizes logic in functions.

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

const count = ref(0);
const doubled = computed(() => count.value * 2);

function increment() {
  count.value++;
}
</script>

<template>
  <p>Count: {{ count }}</p>
  <p>Doubled: {{ doubled }}</p>
  <button @click="increment">Add</button>
</template>
```

Advanced Features:

- **reactive**: `const obj = reactive({ key: value })`.
- **watch**: `watch(count, (newVal) => console.log(newVal))`.
- **provide/inject**: Share data across components.

Edge Cases:

- **Reactivity Loss**: Assigning `ref.value` to a new object.
- **Watch Deep**: `watch(obj, () => {}, { deep: true })`.
- **Setup Syntax**: `<script setup>` simplifies imports.

Cross-Connections:

- **Reactivity**: Core to Vue's UI updates.
- **Pinia**: Composable stores.
- **TypeScript**: Typed `ref`/`reactive`.

Reactivity System

Vue's reactivity uses Proxies for efficient updates.

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

const user = reactive({ name: 'Alice', age: 30 });
const name = ref('Bob');

function update() {
  user.age++;
  name.value = 'Charlie';
}
</script>

<template>
  <p>{{ user.name }}, {{ user.age }} years old</p>
  <p>Hello, {{ name }}!</p>
  <button @click="update">Update</button>
</template>
```

Advanced Features:

- **toRefs**: `const { x } = toRefs(reactiveObj)`.
- **readonly**: Prevent mutations.
- **effectScope**: Group reactive effects.

Edge Cases:

- **Non-Reactive**: Primitives in `reactive` lose reactivity if reassigned.
- **Array Pitfalls**: `arr[0] = x` triggers; `arr.length = 0` may not.
- **Performance**: Deep reactivity on large objects.

Cross-Connections:

- **Computed**: Reactive dependencies.
- **Watch**: Trigger on reactive changes.

- **Pinia**: Reactive state management.

Components

Components are reusable UI blocks.

```
<!-- components/MyButton.vue -->
<script setup>
defineProps(['label']);
defineEmits(['click']);
</script>

<template>
  <button @click="$emit('click')" class="btn">{{ label }}</button>
</template>

<style scoped>
.btn { padding: 8px 16px; background: #42b983; color: white; }
</style>
```

Usage:

```
<script setup>
import MyButton from './components/MyButton.vue';
function handleClick() {
  alert('Clicked!');
}
</script>

<template>
  <MyButton label="Click Me" @click="handleClick" />
</template>
```

Advanced Features:

- **Slots**: `<slot>Default</slot>`.
- **Dynamic Components**: `<component :is="componentName">`.
- **Keep-Alive**: Cache component state.

Edge Cases:

- **Prop Mutation**: Avoid mutating props directly.
- **Event Naming**: Kebab-case in templates (`@my-event`).
- **Slot Scope**: Misaligned scoped slots break rendering.

Cross-Connections:

- **Router**: Components as routes.
- **Pinia**: Share state across components.
- **Testing**: Mount components with Vue Test Utils.

Directives

Directives modify DOM behavior.

```
<script setup>
import { ref } from 'vue';
const isVisible = ref(true);
</script>

<template>
  <div v-if="isVisible">Visible</div>
  <ul>
    <li v-for="n in 3" :key="n">{{ n }}</li>
  </ul>
  <input v-model="isVisible" type="checkbox" />
</template>
```

Custom Directive:

```
<script>
const focus = {
  mounted(el) {
    el.focus();
  }
};
</script>

<template>
  <input v-focus />
</template>
```

Advanced Features:

- **v-bind**: `:class="{ active: isActive }"`.
- **v-on**: `@click.stop.prevent`.
- **v-memo**: Optimize rendering (Vue 3.2+).

Edge Cases:

- **v-if vs. v-show**: `v-if` removes DOM; `v-show` toggles CSS.
- **Key Requirement**: Missing `:key` in `v-for` causes render issues.
- **Directive Hooks**: Incorrect lifecycle usage.

Cross-Connections:

- **Reactivity**: Directives rely on reactive data.
- **Components**: Directives in custom components.
- **Performance**: Optimize directive usage.

Vue Router

Handle SPA navigation.

```
// router/index.js
import { createRouter, createWebHistory } from 'vue-router';
import Home from '../views/Home.vue';

const routes = [
  { path: '/', component: Home },
  { path: '/about', component: () => import('../views/About.vue') },
];

export default createRouter({
  history: createWebHistory(),
  routes,
});
```

```
// main.js
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App).use(router).mount('#app');
```

```
<!-- App.vue -->
<template>
  <router-link to="/">Home</router-link>
  <router-link to="/about">About</router-link>
  <router-view />
</template>
```

Advanced Features:

- **Route Guards**: `beforeEach((to, from) => ...)`.
- **Nested Routes**: Children routes for layouts.
- **Route Meta**: `{ meta: { requiresAuth: true } }`.

Edge Cases:

- **Lazy Loading**: Dynamic imports reduce bundle size.
- **Navigation Failures**: Handle duplicate navigations.
- **Query Params**: Preserve reactive state.

Cross-Connections:

- **Pinia**: Access store in guards.
- **Suspense**: Lazy-load route components.
- **Testing**: Mock router in tests.

State Management with Pinia

Pinia is Vue's official state management library.

```
// stores/tasks.js
import { defineStore } from 'pinia';
```

```
export const useTaskStore = defineStore('tasks', {
  state: () => ({
    tasks: [],
  }),
  getters: {
    completedTasks: (state) => state.tasks.filter(t => t.completed),
  },
  actions: {
    async addTask(task) {
      this.tasks.push({ ...task, id: Date.now() });
      // Simulate API call
      await new Promise(resolve => setTimeout(resolve, 1000));
    },
  },
});
```

Usage:

```
<script setup>
import { useTaskStore } from '../stores/tasks';
const store = useTaskStore();
const task = ref({ description: '', completed: false });

async function add() {
  await store.addTask(task.value);
  task.value = { description: '', completed: false };
}
</script>

<template>
  <input v-model="task.description" />
  <button @click="add">Add Task</button>
  <ul>
    <li v-for="t in store.completedTasks" :key="t.id">{{ t.description }}</li>
  </ul>
</template>
```

Advanced Features:

- **Plugins**: Extend Pinia functionality.
- **Persisted State**: Save to `localStorage`.
- **Reactive State**: Seamless Vue integration.

Edge Cases:

- **Reactivity Loss**: Avoid non-reactive state assignments.
- **Async Actions**: Handle errors in actions.
- **Store Overuse**: Use components for local state.

Cross-Connections:

- **Reactivity**: Pinia leverages Vue's reactivity.

- **Router**: Sync routes with store.
- **Testing**: Mock stores with Vitest.

Advanced Vue Features

Suspense and Lazy Loading

Load components asynchronously.

```
<template>
  <Suspense>
    <template #default>
      <AsyncComponent />
    </template>
    <template #fallback>
      <div>Loading...</div>
    </template>
  </Suspense>
</template>

<script setup>
import { defineAsyncComponent } from 'vue';
const AsyncComponent = defineAsyncComponent(() =>
  import('./components/HeavyComponent.vue')
);
</script>
```

Edge Cases:

- **Error Handling**: Wrap in `ErrorBoundary`.
- **Multiple Suspense**: Nested fallbacks.
- **Hydration**: SSR with Suspense requires care.

Cross-Connections:

- **Router**: Lazy-load routes.
- **Performance**: Reduce initial bundle size.
- **Testing**: Mock async components.

Teleport

Render content outside the component's DOM.

```
<template>
  <button @click="isOpen = true">Open Modal</button>
  <Teleport to="body">
    <div v-if="isOpen" class="modal">
      <p>Modal Content</p>
      <button @click="isOpen = false">Close</button>
    </div>
  </Teleport>
</template>
```

```
<script setup>
import { ref } from 'vue';
const isOpen = ref(false);
</script>

<style scoped>
.modal { position: fixed; background: rgba(0,0,0,0.5); }
</style>
```

Edge Cases:

- **DOM Conflicts**: Multiple teleports to same target.
- **SSR**: Ensure server-side compatibility.
- **Accessibility**: Modals need focus management.

Cross-Connections:

- **Components**: Modal components.
- **Directives**: Enhance with `v-focus`.
- **Testing**: Verify DOM placement.

Render Functions and JSX

Programmatic rendering.

```
// MyComponent.jsx
import { h } from 'vue';

export default {
  props: ['text'],
  render() {
    return h('div', { class: 'custom' }, this.text);
  },
};
```

JSX with Vite:

```
<script>
import { defineComponent } from 'vue';
export default defineComponent({
  props: ['text'],
  render() {
    return <div class="custom">{this.text}</div>;
  },
});
</script>
```

Edge Cases:

- **Performance**: Render functions bypass template compiler.
- **Type Safety**: JSX requires TypeScript setup.

- **Reactivity**: Ensure props are reactive.

Cross-Connections:

- **Components**: Alternative to templates.
- **TypeScript**: Better JSX support.
- **Testing**: Test render output.

TypeScript with Vue

TypeScript enhances Vue with static types.

```
<script setup lang="ts">
import { ref } from 'vue';

interface Task {
  id: number;
  description: string;
  completed: boolean;
}

const task = ref<Task>({ id: 1, description: 'Test', completed: false });

function updateTask(desc: string) {
  task.value.description = desc;
}
</script>

<template>
  <input v-model="task.description" @input="updateTask($event.target.value)" />
  <p>{{ task.completed ? 'Done' : 'Pending' }}</p>
</template>
```

tsconfig.json:

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ESNext",
    "strict": true,
    "jsx": "preserve",
    "moduleResolution": "node",
    "types": ["vite/client", "vue/ref-macros"]
  }
}
```

Advanced Features:

- **DefineProps/DefineEmits**: `defineProps<{ label: string }>()`.
- **Typed Slots**: Type-safe slot props.
- **Volar**: VS Code extension for Vue TS.

Edge Cases:

- **Reactivity Types**: `Ref<T>` vs. `Reactive<T>`.
- **Generic Components**: Complex prop typing.
- **Migration**: Gradual TS adoption in Vue.

Cross-Connections:

- **Pinia**: Typed stores.
- **Router**: Typed routes.
- **Testing**: Type-safe mocks.

Performance Optimization

Optimize Vue apps for speed.

```
<script setup>
import { ref } from 'vue';
const items = ref(Array.from({ length: 1000 }, (_, i) => ({ id: i })));
</script>

<template>
  <div v-for="item in items" :key="item.id" v-memo="[item.id]">
    {{ item.id }}
  </div>
</template>
```

Techniques:

- **Lazy Loading**: Use `defineAsyncComponent`.
- **Tree Shaking**: Vite removes unused code.
- **Virtual Scrolling**: Libraries like `vue-virtual-scroller`.

Tools:

- **Vue Devtools**: Inspect component performance.
- **Lighthouse**: Audit in Chrome DevTools.
- **Vite Analyzer**: Bundle size analysis.

Edge Cases:

- **Reactivity Overhead**: Large reactive objects.
- **Over-Rendering**: Avoid unnecessary updates.
- **Bundle Size**: Split chunks with dynamic imports.

Cross-Connections:

- **Suspense**: Optimize async rendering.
- **Vite**: Fast builds.

- **Testing**: Measure render times.

Security in Vue Apps

Secure Vue apps against vulnerabilities.

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

<template>
  <!-- Safe: Avoid v-html -->
  <p>{{ userInput }}</p>
  <input v-model="userInput" />
</template>
```

Best Practices:

- **XSS Prevention**: Avoid `v-html` with user input; use `sanitize-html`.
- **CSRF**: Use tokens in API requests.
- **Secure APIs**: Validate inputs server-side.

```
// Sanitize user input
import sanitizeHtml from 'sanitize-html';
const safeHtml = sanitizeHtml(dirtyHtml);
```

Edge Cases:

- **DOM Injection**: `v-html` with unsanitized input.
- **Third-Party Libs**: Audit dependencies (`npm audit`).
- **CORS**: Restrict API access.

Cross-Connections:

- **Pinia**: Secure state mutations.
- **Router**: Protect routes.
- **Testing**: Verify sanitization.

Testing Vue Apps

Vitest

Unit test Vue components.

```
// tests/TaskItem.test.js
import { mount } from '@vue/test-utils';
import { describe, it, expect } from 'vitest';
import TaskItem from '../components/TaskItem.vue';

describe('TaskItem', () => {
  it('renders task description', () => {
```

```
    const wrapper = mount(TaskItem, {
      props: { task: { id: 1, description: 'Test', completed: false } },
    });
    expect(wrapper.text()).toContain('Test');
  });

  it('emits toggle event', async () => {
    const wrapper = mount(TaskItem);
    await wrapper.find('input').trigger('click');
    expect(wrapper.emitted().toggle).toBeTruthy();
  });
});
```

Run:

```
npm run test
```

Vue Test Utils

Mount and interact with components.

```
import { mount } from '@vue/test-utils';
import MyButton from '../components/MyButton.vue';

const wrapper = mount(MyButton, { props: { label: 'Click' } });
await wrapper.trigger('click');
expect(wrapper.emitted('click')).toHaveLength(1);
```

Cypress

End-to-end testing.

```
// cypress/e2e/tasks.cy.js
describe('Task App', () => {
  it('adds a task', () => {
    cy.visit('/');
    cy.get('input').type('New Task');
    cy.get('button').contains('Add').click();
    cy.contains('New Task').should('be.visible');
  });
});
```

Run:

```
npx cypress run
```

Advanced Features:

- **Mocking**: Mock Pinia with `createTestingPinia`.
- **Snapshots**: Compare DOM output.
- **Accessibility**: Test with `cypress-axe`.

Edge Cases:

- **Async Rendering**: Await `wrapper.vm.$nextTick()`.

- **Router Testing**: Mock `useRouter`.
- **Coverage Gaps**: Combine unit and E2E tests.

Cross-Connections:

- **Pinia**: Test store actions.
- **Router**: Test navigation.
- **TypeScript**: Type-safe tests.

Tooling and Ecosystem

Vite

Fast build tool for Vue.

```
// vite.config.js
import { defineConfig } from 'vite';
import vue from '@vitejs/plugin-vue';

export default defineConfig({
  plugins: [vue()],
  test: {
    environment: 'jsdom',
    coverage: { provider: 'v8' },
  },
});
```

Vue CLI

Alternative for legacy projects.

```
npm install -g @vue/cli
vue create my-app
```

Nuxt.js

SSR and static site generation.

```
// nuxt.config.js
export default {
  modules: ['@pinia/nuxt'],
  buildModules: ['@nuxt/typescript-build'],
};
```

Advanced Features:

- **HMR**: Hot module replacement in Vite.
- **SSG**: Static site generation with Nuxt.
- **Plugins**: Extend Vite/Nuxt functionality.

Edge Cases:

- **Build Errors**: Misconfigured plugins.

- **SSR Hydration**: Mismatched client/server DOM.
- **Dependency Bloat**: Optimize with `vite-plugin-analyzer`.

Cross-Connections:

- **Pinia**: Nuxt auto-imports stores.
- **Router**: Nuxt handles routing.
- **Testing**: Vite's test runner.

Sample Project: Task Management App

This project builds a **task management SPA** using Vue 3, Vue Router, Pinia, TypeScript, Vite, and Vitest, demonstrating modern Vue development.

Project Structure:

```
task-manager/
├── src/
│   ├── assets/
│   │   └── styles.css
│   ├── components/
│   │   ├── TaskItem.vue
│   │   └── TaskForm.vue
│   ├── views/
│   │   ├── Home.vue
│   │   └── Tasks.vue
│   ├── stores/
│   │   └── tasks.ts
│   ├── router/
│   │   └── index.ts
│   ├── App.vue
│   ├── main.ts
│   └── types.ts
├── tests/
│   ├── TaskItem.test.ts
│   └── e2e/
│       └── tasks.cy.ts
├── public/
│   └── favicon.ico
├── vite.config.ts
├── tsconfig.json
├── package.json
└── README.md
```

package.json:

```
{
  "name": "task-manager",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "test": "vitest run",
    "test:watch": "vitest",
    "cypress": "cypress run"
```

```
  },
  "dependencies": {
    "vue": "^3.5.0",
    "vue-router": "^4.4.0",
    "pinia": "^2.2.0",
    "sanitize-html": "^2.13.0"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^5.1.0",
    "vite": "^5.4.0",
    "typescript": "^5.6.0",
    "vitest": "^2.1.0",
    "@vue/test-utils": "^2.4.0",
    "cypress": "^13.15.0",
    "jsdom": "^25.0.0",
    "vue-tsc": "^2.1.0"
  }
}
```

types.ts:

```
export interface Task {
  id: number;
  description: string;
  completed: boolean;
  createdAt: Date;
}
```

stores/tasks.ts:

```
import { defineStore } from 'pinia';
import type { Task } from '../types';

export const useTaskStore = defineStore('tasks', {
  state: () => ({
    tasks: [] as Task[],
  }),
  getters: {
    completedTasks: (state): Task[] => state.tasks.filter(t => t.completed),
    pendingTasks: (state): Task[] => state.tasks.filter(t => !t.completed),
  },
  actions: {
    async addTask(description: string) {
      const task: Task = {
        id: Date.now(),
        description,
        completed: false,
        createdAt: new Date(),
      };
      this.tasks.push(task);
      // Simulate API call
      await new Promise(resolve => setTimeout(resolve, 500));
    },
    toggleTask(id: number) {
      const task = this.tasks.find(t => t.id === id);
      if (task) task.completed = !task.completed;
```

```
    },
  },
});
```

router/index.ts:

```
import { createRouter, createWebHistory } from 'vue-router';
import Home from '../views/Home.vue';
import Tasks from '../views/Tasks.vue';

const routes = [
  { path: '/', component: Home },
  { path: '/tasks', component: Tasks },
];

export default createRouter({
  history: createWebHistory(),
  routes,
});
```

components/TaskItem.vue:

```
<script setup lang="ts">
import type { Task } from '../types';
defineProps<{ task: Task }>();
defineEmits<{
  (e: 'toggle', id: number): void;
}>();
</script>

<template>
  <li :class="{ completed: task.completed }">
    <input type="checkbox" :checked="task.completed" @change="$emit('toggle',
task.id)" />
    <span>{{ task.description }}</span>
    <small>{{ task.createdAt.toLocaleDateString() }}</small>
  </li>
</template>

<style scoped>
.completed { text-decoration: line-through; color: #888; }
li { display: flex; gap: 8px; align-items: center; }
</style>
```

components/TaskForm.vue:

```
<script setup lang="ts">
import { ref } from 'vue';
import { useTaskStore } from '../stores/tasks';
import sanitizeHtml from 'sanitize-html';

const store = useTaskStore();
const description = ref('');
const error = ref('');

async function addTask() {
```

```
  const cleanDesc = sanitizeHtml(description.value, { allowedTags: [] });
  if (!cleanDesc.trim()) {
    error.value = 'Description required';
    return;
  }
  try {
    await store.addTask(cleanDesc);
    description.value = '';
    error.value = '';
  } catch (err) {
    error.value = 'Failed to add task';
  }
}
</script>

<template>
  <form @submit.prevent="addTask">
    <input v-model="description" placeholder="New task" aria-label="Task description"
/>
    <button type="submit">Add</button>
    <p v-if="error" class="error">{{ error }}</p>
  </form>
</template>

<style scoped>
form { display: flex; gap: 8px; }
.error { color: red; }
</style>
```

views/Tasks.vue:

```
<script setup lang="ts">
import TaskItem from '../components/TaskItem.vue';
import TaskForm from '../components/TaskForm.vue';
import { useTaskStore } from '../stores/tasks';
const store = useTaskStore();
</script>

<template>
  <div>
    <h2>Tasks</h2>
    <TaskForm />
    <h3>Pending</h3>
    <ul>
      <TaskItem
        v-for="task in store.pendingTasks"
        :key="task.id"
        :task="task"
        @toggle="store.toggleTask(task.id)"
      />
    </ul>
    <h3>Completed</h3>
    <ul>
      <TaskItem
        v-for="task in store.completedTasks"
        :key="task.id"
```

```
        :task="task"
        @toggle="store.toggleTask(task.id)"
      />
    </ul>
  </div>
</template>

<style scoped>
div { max-width: 600px; margin: 0 auto; }
</style>
```

views/Home.vue:

```
<template>
  <h1>Welcome to Task Manager</h1>
  <p><router-link to="/tasks">View Tasks</router-link></p>
</template>
```

App.vue:

```
<script setup>
import { RouterLink, RouterView } from 'vue-router';
</script>

<template>
  <nav>
    <RouterLink to="/">Home</RouterLink> |
    <RouterLink to="/tasks">Tasks</RouterLink>
  </nav>
  <main>
    <RouterView />
  </main>
</template>

<style scoped>
nav { text-align: center; padding: 16px; }
</style>
```

main.ts:

```
import { createApp } from 'vue';
import { createPinia } from 'pinia';
import App from './App.vue';
import router from './router';
import './assets/styles.css';

const app = createApp(App);
app.use(createPinia());
app.use(router);
app.mount('#app');
```

assets/styles.css:

```
body {
  font-family: Arial, sans-serif;
  line-height: 1.6;
  margin: 0;
```

```
  padding: 0;
}
button {
  background: #42b983;
  color: white;
  border: none;
  padding: 8px 16px;
  cursor: pointer;
}
input {
  padding: 8px;
  border: 1px solid #ccc;
}
```

tests/TaskItem.test.ts:

```
import { mount } from '@vue/test-utils';
import { describe, it, expect } from 'vitest';
import TaskItem from '../components/TaskItem.vue';
import type { Task } from '../types';

describe('TaskItem', () => {
  const task: Task = {
    id: 1,
    description: 'Test',
    completed: false,
    createdAt: new Date(),
  };

  it('renders task description', () => {
    const wrapper = mount(TaskItem, { props: { task } });
    expect(wrapper.text()).toContain('Test');
  });

  it('emits toggle event', async () => {
    const wrapper = mount(TaskItem, { props: { task } });
    await wrapper.find('input').trigger('click');
    expect(wrapper.emitted('toggle')).toEqual([[1]]);
  });
});
```

tests/e2e/tasks.cy.ts:

```
describe('Task App', () => {
  it('adds a task', () => {
    cy.visit('/tasks');
    cy.get('input').type('New Task');
    cy.get('button').contains('Add').click();
    cy.contains('New Task').should('be.visible');
  });
});
```

vite.config.ts:

```
import { defineConfig } from 'vite';
import vue from '@vitejs/plugin-vue';
```

```
export default defineConfig({
  plugins: [vue()],
  test: {
    environment: 'jsdom',
    coverage: { provider: 'v8' },
  },
});
```

tsconfig.json:

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ESNext",
    "strict": true,
    "jsx": "preserve",
    "moduleResolution": "node",
    "types": ["vite/client", "vue/ref-macros", "vitest/globals"],
    "allowSyntheticDefaultImports": true,
    "baseUrl": ".",
    "paths": {
      "@/*": ["src/*"]
    }
  },
  "include": ["src/**/*", "tests/**/*"]
}
```

Features Demonstrated:

- **JavaScript**: Modern ES2025, async/await, modules.
- **Vue.js**: Composition API, reactivity, components, directives.
- **Routing**: Vue Router for SPA navigation.
- **State Management**: Pinia with reactive stores.
- **TypeScript**: Typed components, props, and stores.
- **Testing**: Vitest for unit tests, Cypress for E2E.
- **Security**: Sanitized inputs with `sanitize-html`.
- **Performance**: Lazy-loaded routes, memoized rendering.
- **Tooling**: Vite for fast builds, Volar for IDE support.

Running the Project:

```
npm create vue@latest
cd task-manager
npm install
npm run dev
```

Test:

```
npm run test
npx cypress run
```

Sample Usage:

- Navigate to `http://localhost:5173`.
- Add tasks via the form on `/tasks`.
- Toggle task completion with checkboxes.
- View pending and completed tasks separately.

Edge Cases Handled:

- **Reactivity**: Proper `ref`/`reactive` usage.
- **Async**: Error handling in async actions.
- **Security**: Sanitized user input.
- **Type Safety**: TypeScript interfaces and props.
- **Accessibility**: ARIA labels, focus management.
- **Performance**: Optimized rendering with `v-memo`.

Resources

- **Official Docs**:
  - JavaScript (MDN),
  - Vue.js,
  - Vue Router,
  - Pinia,
  - Vite,
  - Nuxt.js.
- **Tutorials**:
  - Vue Mastery,
  - Vueschool,
  - JavaScript.info.
- **Community**:
  - Stack Overflow,
  - Reddit r/vuejs,
  - Vue Discord.
- **Tools**:

    - VS Code,
    - Volar,
    - Vue Devtools,
    - Vitest,
    - Cypress.
- **Libraries**:
    - NPM,
    - Awesome Vue.
- **Books**:
    - *JavaScript: The Definitive Guide* (O'Reilly),
    - *Fullstack Vue* (Newline),
    - *Vue.js 3 By Example* (Packt).
- **Security**:
    - OWASP XSS,
    - Vue Security.

This guide and sample project provide a comprehensive foundation for mastering JavaScript and Vue.js, from basic syntax to advanced reactive UI development, with practical applications and deep insights into Vue's ecosystem.